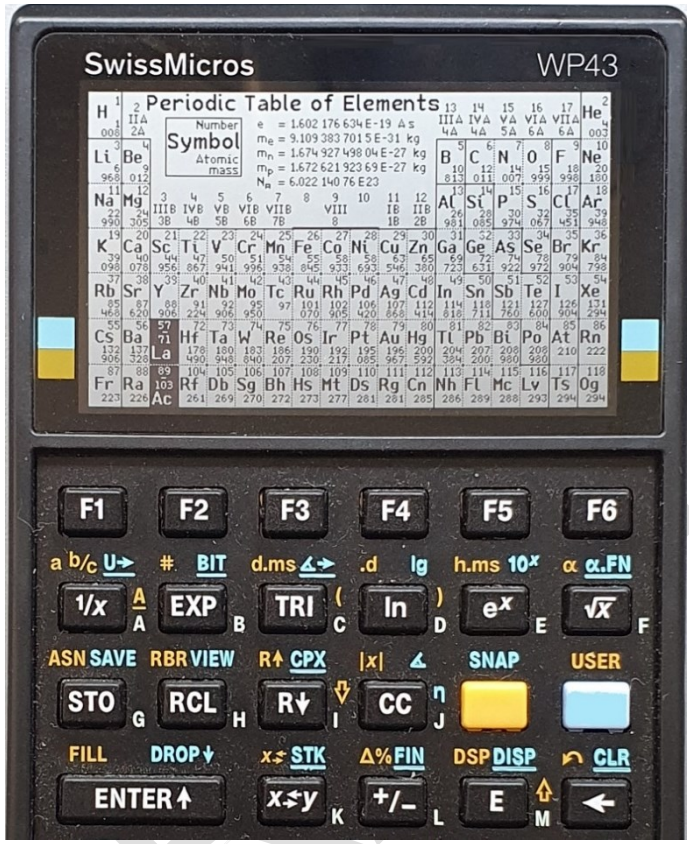


WP43 REFERENCE MANUAL

This manual supplements the *WP43 Owner's Manual and Problem Solving Guide*. **(***WP43* is a free scientific software for *DM42* calculator hardware built by *SwissMicros*. You can redistribute that software and/or modify it under the terms of the *GNU General Public License* as published by the *Free Software Foundation*, either version 3 of the *License*, or (at your option) any later version. *WP43* is published and distributed in the hope that it will be useful, but without any warranty; without even the implied warranty of merchantability or fitness for a particular purpose. Please see *http://www.gnu.org/licenses/* for more details.**)**

This manual is preliminary still; it will change while we develop *WP43* in course of this project. We reserve the right to do so at any time. The fundamental principles of *WP43* will remain constant, however. You can stay informed by watching *https://gitlab.com/rpncalculators/wp43*.

Copyright © 2015 - 2025 Walter Bonin, Radebeul, Germany

All rights reserved. No part of this manual may be copied, displayed, extracted, reproduced, utilized, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical or otherwise including but not limited to photocopying, recording, or scanning without the prior written consent of the author. As long as it is just a draft, this manual may be downloaded but not changed in any way. The locations highlighted cyan are open issues – information is missing there or needs further discussion or investigation to be determined. Any contributions in these matters are appreciated.

HP is a registered trade mark of *Hewlett-Packard*. *Excel* and *WordPad* are applications of *Microsoft*.

The drawing on p. 185 was kindly supplied by *SwissMicros*. The plots in *Appendix I* (unless specified otherwise locally) and the pictures on pp. 179 (bottom left) and 323 are based on material found in *Wikipedia*. Other sources may be stated locally. The pictures, diagrams, and graphics beyond were created by the author.

Over 100 internet addresses and links are specified as found and verified on 2025-05-11. Please note such addresses may change without notice at any time.

This manual is published in English since it became the *lingua franca* of our time (after Greek, Latin, and French in Europe) – using it we can reach the maximum number of readers without further translations. I apologize to the people of other languages and inserted some 'translator's notes' where applicable.

And if you find some barbed remarks in the text, they just reflect the author's impressions watching this world for seven decades so far. No personal offense intended but over-confident people might get some.

Printed in the USA

ISBN-13: 978-172950106-1
ISBN-10: 172950106-0

WP43 would neither have been created without our love for *Classics, Woodstocks, Stings, Spices, Nuts, Voyagers*, and *Pioneers* nor without our ability to imagine better products on a way towards a better world. Thus we want to quote what was printed in *HP* pocket calculator manuals until 1980, so it will not fade:

"The success and prosperity of our company will be assured only if we offer our customers superior products that fill real needs and provide lasting value, and that are supported by a wide variety of useful services, both before and after sales."

Statement of Corporate Objectives
Hewlett-Packard

DRAFT

Just in Case You Do Not Have Your Own *WP43* Calculator yet …

A pilot batch of *WP43* calculators was made on *SwissMicros'* production line in China in the summer of 2022. These were sold out in few weeks – thus, please turn to *https://www.swissmicros.com/* expressing your wishes. Maybe they will listen to their customers.

Alternatively, you can create a *WP43* most easily by flashing a *DM42* you own – this way will cost you next to nothing but some key stickers and an overlay will become inevitable. Please consult *App. F* of *this manual* for what and how to do.

The *WP43* software package includes a free full size *Simulator* (see *App. E*), so you can check it out on your computer before buying any hardware. And later on, you may test and edit your programs there as well before sending them to your *WP43* calculator. Function sets are 99% identical on both platforms.

For the following, we assume you hold your own *WP43* in your hands – or have access to its *Simulator* at least.

TABLE OF CONTENTS

Welcome! **11**

Print Conventions and Common Abbreviations 12

Section 1: The Index of Items (*IOI*) **14**

Commands, Menus, and Further Item Names Reserved 15

0 – 9 20
A 20
B 24
C 26
D 33
E 38
F 40
G 43
H 45
I 47
J 50
K 51
L 53
M 57
N 65
O 68
P 68
Q 72
R 72
S 80
T 89
U 92
V 93
W 94
X 95

DRAFT

Y 98
Z 99
A, α 99
β 100
Γ , γ 101
Δ, δ 101
ε 101
ζ 102
λ 102
μ 102
Π, π 102
Σ, σ 103
Φ 104
χ 105
ψ 105
ω 105
(, [, +, -, ×, /, ^ 105
→, | 106
% 109
√, ∫, x̄, 🖶, # 109
Constants 113
System Flags 127
Tests 131
Nonprogrammable Commands and Keys 132

Section 2: Menus and Catalogs 133

One to Find and Rule Them All – the CATALOG 133
Accessing Cataloged Items Rapidly 136
Further Menus and Their Contents 138
Unit Conversions 150

DRAFT

Section 3: Calling and Executing Operations **163**
Using XEQ for Executing Operations 163
Command Parameter Input and Closing It 164
Alphanumeric Input in **X** and Closing It 166
Operations Requiring Trailing Parameters 170
Operations Allowing For Indirect Addressing 172
Operations Changing Data Types 173
Operations in Equations 175

Appendix A: Hardware **178**
Off-Images 184
Overlays 184

Appendix B: Memory Management **187**
Data Types 187
What Will Happen when Changing Word Size? 190
SAVE and LOAD… 190
Range and Representation of Real Numbers 191
Limitations 195
Special Results 197
Program Step Sizes 202
Program Files 205

Appendix C: Messages and Error Codes **206**

Appendix D: Differences to the Function Sets of *HP-42S, HP-16C, HP-21S*, and *WP 34S* **213**
Corresponding Operations on *HP-42S* 213
Corresponding Operations on *HP-16C* 219
Corresponding Operations on *HP-21S* 221
Corresponding Operations on *WP 34S* 223

DRAFT

New Commands on Your *WP43* 227
Reference Literature 237

Appendix E: Emulating a *WP43* on Your *PC* 239

How to Exchange Programs Between Your *WP43* and Your *PC* 241

Appendix F: Flashing & Updating Your *WP43* 244

How to Create Your *WP43* by Flashing a *DM42* 244
How to Update Your *WP43* 246

Appendix G: Troubleshooting Guide 248

First a Few Words About Coin Cells 248
Calculator Freeze 249
Exception 250
Fresh Quality Battery Constantly Low 250
Individual Keys not Reacting Properly 251
Keymap Trouble 251
Other Trouble 252

Appendix H: Historic Remarks and Caveats for Younger Users (What Grandpa Still Knew) 253

Keys and Keyboards 254
Reverse Polish Notation 255
Why Does ENTER↑ Work the Way It Does? And Why Are there Automatic Stack Lifts and Drops? 256
How Many Stack Levels Are Necessary? 259
How Many Stack Levels Shall Be Displayed? 261
Precision and Accuracy of Calculations with Real Numbers 261
About the Classic *Mach Number* Formula 264

DRAFT

Appendix I: Advanced Mathematical Functions and Tasks 267

Number Generating Functions 267
Statistical Distribution Functions (*PMF*, *PDF*, *CDF*, etc.) 269
More Statistical Formulas, also for Curve Fitting 278
Curve Fitting Models Provided 286
The Pedestrian's Measurement System Analysis (*MSA*) 295
Error Propagation in Calculations 298
Solving Differential Equations (1) 300
Solving Differential Equations (2) 305
Orthogonal Polynomials 311
Even More Mathematical Functions 316

Appendix J: Information for Advanced Users 325

Recursive Programming 325
A Few Words About Factorizing 325
A Few Words About the *Solver* 326
More Than a Few Words About the *Integrator* 327
Differentiating Real Functions Using Complex Numbers 331
A Few Words About Random Numbers 332
A Few Words About Uncertainty, Significance, and Confidence 332
A Few Words About Curve Fitting 333
A Few Words About *TVM* 333
Translation of *WP43* Characters Deviating from *UTF-8* 334
Where to Find Programs Readily Available? 337

Appendix K: The Most Honorable Members of the Pilot Run 338

Appendix L: Making a *WP43* Stand for Less Than $2 Cost 339

Appendix M: Release Notes 340

Appendix N: Index of Names, Calculators, Companies, etc. 349

DRAFT

WP43 Quick Reference Guide **Q-1**

ADDRESS SPACE 1
DATA TYPES 2
USING MENUS 3
MODES 4
DISPLAY FORMATS 4
PROGRAMMING 5
EXECUTING FUNCTIONS AND PROGRAMS 6
CLEARING AND DELETING 7
MATRIX OPERATIONS 8
PROBABILITY 10
STATISTICS 11
ADVANCED OPERATIONS 12
OPERATIONS ON SHORT INTEGERS 14
OPERATIONS ON TEXT STRINGS 15
USER WRITEABLE SYSTEM FLAGS 16

Background Facts and Considerations **B-1**

Accessing Items 1
Angles 7
Calculation Internals 8
Character Sets 8
Complex Notation, Storage, and Calculations Close to Infinity 14
Display Segmentation 16
Echo and Fallback 17
Equations 17
Execution Times 18
Functionality of ENTER↑ 19
Names of User Defined Items 19
Number Range 19

DRAFT

Precision and Accuracy 20
Prefixes 21
Random Numbers 21
RPN vs. RPL or Vice Versa 21
Sorting in Detail 22
Stack Lift Disabling Functions 27
Stack Size 28
Structured Programming 28
UNDO 29

For Your Convenience B-31

DRAFT

WELCOME!

This reference volume of the *WP43* documentation supplements your *WP43 Owner's Manual* with comprehensive information about each and every *item* (i.e. command, *menu*, *catalog*, browser, application, conversion, *variable*, constant, *system flag*, *global label*, and *character*) provided in your *WP43*. With these two manuals, you have got all the user information about *WP43* in your hands in merely two files.[1]

Section 1 with the *Index of Items* and dedicated chapters, e.g., about *system flags* takes more than a quarter of this manual.

Section 2 presents the structures and contents of all the *menus* and *catalogs* provided.

Section 3 shows further access methods to operations and lists all functions requiring parameters, allowing for indirect addressing, changing *data types*, and being accessible in equations.

Appendices A – N cover special topics like error messages, the *WP43 Simulator*, and *WP43* firmware updates. And there are lots of information more – please check the *Table of Contents* for details.

You find a 16-page ***Quick Reference Guide*** close to the end of this manual, followed by background information about this project, and some helpful prints for your convenience at the very end of this volume.

And here is the inevitable
WARNING: Read the WARNINGS in your *WP43 Owner's Manual*, pp. 16f! Else you are only allowed to look at your *WP43*. And there is one great WARNING in *Appendix H*, though concerning vintage calculators.

Enjoy!

Walter

[1] This manual might apply to *WP43* offsprings like *C43, C47*, *R47*, etc. to some extent. It can't replace dedicated manuals for those, however, which are yet to come one day.

DRAFT

Print Conventions and Common Abbreviations

- Standard text font in our manuals is *Arial*. Emphasis is added by underlining or **bold** printing. COMMANDS, MENUS, predefined VARIABLES and SYSTEM FLAGS (SET or ~~CLEAR~~) are generally called by their *names*, printed capitalized in running text.

  Examples are printed in green, quoted text in blue (as well as translator's notes). Bold italic letters (like ***n***) denote *variables*, bold regular *characters* (like **b**) constant sample values (e.g., specific labels, numbers, or symbols).

  Specific *terms, titles, trademarks, names*, or *abbreviations* are printed in italics, *hyperlinks* in blue underlined italics. Each link will beam you to its target – for best use in print, it will generally refer to a page number, the *Table of Contents*, or an external address fully specified.

- *Times New Roman* regular letters are used for unit symbols and mathematical functions, italics for *unit names* in running text.

  Bold capitals denote **REGISTERS**, lower case bold italics their ***contents***.[2] E.g., *register* **Y** contains the value ***y***, and **R45** contains ***r45***. *Stack* contents are generally quoted in order [***x***, ***y***, ***z***, …].

- This KEY font (created by *Luiz Vieira* of Brazil) denotes calculator KEYS, incl. softkeys in general. For shifted operations like GTO or SF, the respective colors are used. `Text` and numeric outputs (like 1.234×10^{-56} or $7{,}089\cdot10^{12}$) are printed as seen on your calculator screen.

- *Courier* is used for `filenames` and describing `numeric formats`.

- Regarding mathematical notation, we follow *ISO 80000-2* to comply with international standards. For the manuals, we chose decimal points, multiplication crosses, and division slashes – you may choose decimal commas, multiplication dots, and division colons on your *WP43* as well, of course.

All this holds unless stated otherwise locally.

The following abbreviations are used throughout this manual. They are introduced here for your reference (a few more may be used and explained locally):

[2] We kept the term *register* for the space where a particular object is stored, although the actual size of such a *register* may vary widely following the size of the object stored therein.

DRAFT

OM = Owner's Manual (referring to the *WP43 OM* unless another model is mentioned explicitly); *OMS **n*** refers to its *Section **n***.
1D, 2D, 3D = one-dimensional, two-dimensional, three-dimensional.
ADM = angular display mode (see *OMS 2*).
AIM = alpha input mode (see *OMS 2*).
App. = Appendix.
ASL = automatic stack lift (see *OMS 1*).
CDF = cumulated distribution function (see *OMS 2*).
DT = data type (see *App. B*).
FD = the *USB FAT disk* (see *App. A*).
FM = flash memory (incl. the *FD*, see *App. A* and *OMS 3*).
f. = footnote.
GP = general purpose (see *OMS 1*).
HP = *Hewlett-Packard.*
IOI = *Index of Items* (see pp. 20ff).
L.R. = linear regression (covering *OLS* and *TLS*, see *App. I*).
MSA = measuring system analysis (see *App. I*).
OH = Owner's Handbook.
OLS = ordinary least squares regression (compare *TLS* = total ~).
PC = computer of arbitrary kind and brand.
PCB = printed circuit board.
PDF, PMF = probability density (or mass) function (see *OMS 2*).
PEM = program-entry mode (see *OMS 3* and *RUM*).
PET = Programming Examples and Techniques.
PP = program pointer (see *OMS 3*).
px = pixels.
RAM = random access memory, allowing read and write operations.
RPN = reverse Polish notation (see *OMS 1*).
RoV = register or variable.
RUM = run mode (see *OMS 1* and cf. *PEM*).
SD = standard deviation (see *App. I*).
SDC = startup default configuration.
SM = *SwissMicros.*
SRS = subroutine return stack (see *App. B*).
TN = translator's note (*TNG* = *TN* for readers speaking German).
TVM = *Time Value of Money* – a preprogrammed application featured by all financial *HP* calculators since 1972 (see *OMS 5*).

DRAFT

SECTION 1: THE INDEX OF ITEMS (*IOI*)

All the *items* (cf. *OMS 1*) provided on your *WP43* are explained in this section. Almost all of them are listed in the table starting on p. 20 with their *names* (as they are displayed in programs) and the way to call them. Most of them are picked from *menus*; then we recorded the respective *menu*, its *view*, and the *label* shown therein (printed on gold or blue if 🟨 or 🟦 is required for accessing it); we are confident you will find the corresponding *softkey*. CAT. abbreviates CATALOG.

After this table, the *constants* and *system flags* provided are explained in separate chapters comprehensively. *Menus* are covered in *Section 2*.

Each *item* on your *WP43* is identified by its unique *name* of up to 7 *characters* – it may be accessible under one or more different *labels* printed on the bezel or displayed in *menus*, featuring less or more *characters* than its *name* (see some unit conversions, for example). Such *labels* are not required to be unique; so one particular *label* may point to or call various *items*, depending on its context.

Since there is no agreed standard for character sorting, we had to invent our own: sorting (*names*, for instance) works in the following order on your *WP43*:

␣ 0…9 Aa…Zz Aα…Ωω ( ) [ ] { } + ± - × / ^
, . ! ? : ; ' " « » · @ _ ~ → ← ↑ ↓ ¬ ∧ ∨ & |
≪ < ≤ ≡ := = ≈ ≙ ≜ ≠ ≥ > ≫ % $ € £ ¥
√ ∞ ∫ ⊙ ⊕ ⊾ ∡ ⊥ 🕓 ⌛ 🖶 🔋 # UK US

Accented letters follow their parents, as do superscripts and subscripts.[3] The same applies to Euler's e, i, and the ℓ for *liter*.

There are some *labels* printed on the keyboard of your *WP43* which will execute differently depending on the context you access them:

[3] The two *characters* printed on grey background here are only indirectly accessible for the user. Find all *characters* created for *WP43* in *Background Facts and Considerations* near the end of this manual.

Label	Context, condition	Functionality
a b/c	Fraction display off	Displays ***x*** as a *proper fraction*
	Fraction display on	Toggles *proper* and *improper fraction* display
#	Input open	Follows a short integer, preceding its base
	Closed ***x***	If followed by a number then calls →INT ***base*** if followed by D then calls →INT **10** if followed by F then calls FP if followed by H then calls →INT **16** if followed by I then calls IP
d.ms	Input open	Closes input of an *angle* in *sexagesimal degrees*
	Closed ***x***	Calls D.MS
.d	Input open	Closes input of a *date*
	Closed ***x***	Calls →REAL
h.ms	Input open	Closes input of a sexagesimal *time*
	Closed ***x***	Calls →H.MS
CC, ENTER↑, ⬅, XEQ, ▲, ▼, R/S, EXIT	Various	Please see the *OM*, *App. 1*, for comprehensive explanation of the functionalities of these eight context sensitive keys

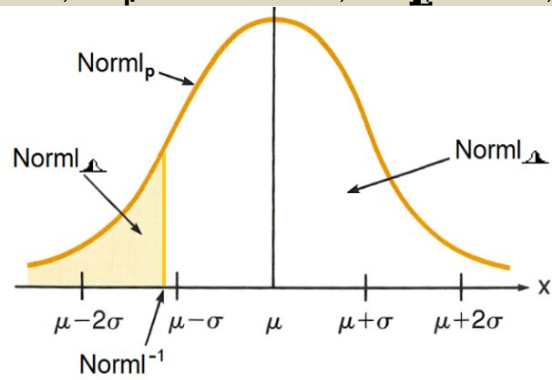
Commands, Menus, and Further Item Names Reserved

In principle, *WP43* operations work as the corresponding ones on the *WP 34S* where applicable (see *App. D*). Referring to vintage *HP* calculators, most functions work as they did on the *HP-42S*, bit and *integer* functions as on the *HP-16C*, unless specified otherwise.

369 operations listed in the *IOI* are new compared to *HP's RPN* pocket calculators.[4] If known from prior such calculators but deviating

[4] We neither compared *HP's RPL* calculators nor the *HP Prime*. They exceed the realm of shirt pocket calculators obviously.

functionally on *WP43,* their remarks are printed on light red. Functions operating on the accumulated statistical data are highlighted light blue. For any probability distribution **XY**, **XY$_p$** labels its *PDF*, **XY$_\Delta$** its *CDF*, **XY$_\Delta$** its error probability, and its quantile function is called **XY^{-1}** (as pictured here for NORML). Commands working exclusively on the calculator hardware (i.e. not on the *Simulator*) have their remarks printed on gold.



In the first column of the *IOI*,

- operations allowed in equations are printed on yellow,
- those whose results are reflected in the *status bar* on grey,
- those not allowed in programming (*PEM*) on violet background.
- Commands asking you for your confirmation are printed in red.
- You can UNDO every operation but those printed in red or orange.

For the vast majority of operations, their respective remarks start with the number of keystrokes needed to call them (enclosed in brackets [ ]). In most cases then follows a number representing …

(0) functions without any effects on the *stack* (like mode setting);

(1) *monadic functions*,

(2) *dyadic functions*, and

(3) *triadic functions* as defined in *OMS 1*; and

(−1) or (−2) functions pushing one or two objects on the *stack.*

Note that some functions overwrite two *stack* levels: e.g., →POL and →REC, as you may have expected.[5]

[5] The *HP-42S* does that also with statistical functions returning two values – while the *Spices* (e.g., *HP-34C*) and *Voyagers* (e.g., *HP-15C*) push both results on the *stack*

Operation *parameters* will be taken from the lowest *stack register*(s) unless mentioned explicitly in column 2 of the *IOI* – then they will follow the command. Most parameters of probability distributions shall be given in *registers* **I**, **J**, and **K** as specified.

We show below three **examples** of the parameter notation used in the *IOI*. Let's assume **R12** contains **15.67** here, i.e. ***r12*** = **15.67** generally.

1. ***n*** represents an arbitrary *integer number* which must be keyed in directly, while

   n represents such a number which may be specified indirectly via an *RoV* as well (as shown in the addressing tables in *OMS 1*); and

   n stands for the respective number itself;

   Example: RSD **12** rounds ***x*** to 12 significant digits, while
   RSD **→12** rounds ***x*** to 15 significant digits.

2. ***r*** (or ***s***) represents an arbitrary *register* address or *variable name* which must be keyed in directly or picked from a *menu*, while

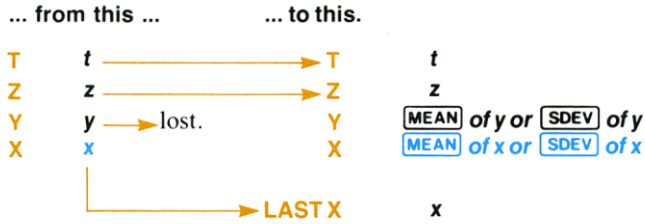
   r (or ***s***) represents such an address or *name* which may be specified indirectly as well; and

   r (or ***s***) stands for the contents of the *RoV* specified – ***r*** or ***s*** may be used as an address itself.

---

instead, as you expect from *RPN* calculators. The picture shows what the *HP-55*, *19C/29C*, *67/97*, *41C*, and *42S* do there:

The illustration below shows what happens in the stack when you execute [MEAN] or [SDEV]. The contents of the stack registers are changed…



As far as we know, *HP* did not give any reason for this deviation from plain *RPN* logic until today. In our opinion this is not sensible, so we stick to the paradigm as implemented on *Spices* and *Voyagers, then WP 34S* and *WP 31S* also for *WP43*.

DRAFT

Example: STO **12** stores x into **R12**, while
STO →**12** stores x into **R15**.

3. ***label*** represents an arbitrary program *label* which must be keyed in directly or picked from a *menu*, while

   label represents such a *label* which may be specified indirectly (as shown in the addressing table in *OMS 3*); and

   label stands for the respective *label* itself, regardless of the way it was specified.

Example: GTO **12** goes to *local label* **12**, while
GTO →**12** goes to *local label* **15**.

Note that for every command ***XYZ*** requiring one trailing parameter, you can enter ***XYZ*** → **X** and it will take its parameter from **X** instead, like a good old traditional *RPN* command.

The *data types* (*DT*) a particular function operates on are listed in { } in the *IOI* under *Remarks* if there are any restrictions (see *App. B* for all available *DTs*). Many bit and *integer* functions operate sensibly on *short integers* (*DT 10*) only. Other functions may work with more *DTs*. Functions stating *DT 8* or *9* operate on entire *real* or *complex matrices* as a whole while those stating *DT 8** or *9** operate on the individual *matrix elements* instead (see *OMS 2*). Wherever an operation returns a *DT* differing from its input, its output *DT* is listed as well.[6]

ASL is enabled after each and every command except CLX, ENTER↑, Σ+, Σ−, M.EDI, M.EDIN, M.←, M.↑, M.→, and M.↓ – input immediately following one of these ten operations will overwrite x instead of pushing it on the *stack*. Reasons are given in *App. H*.

[6] This applies to conversions like °C→°F, for instance: For *real* input (*DT 2*), output will stay *real*. For *integer* input, however, output will become *real*. See there.

Some functions operating on *long integers* (*DT 1*) will return either such integers or *reals*, depending on input. E.g., **27** $\sqrt[3]{x}$ will return 3, but **28** $\sqrt[3]{x}$ will return a *real number* although input is a *long integer* as well. $\sqrt[3]{x}$ operating on *short integers* (*DT 10*) will return *DT 10* always. Cf. *OMS 2*, *Data Types Supported* and *Integers: Summary of Functions*.

Functions which don't work yet for whatever reasons or were not checked yet are marked in column 3. This also applies to the *DTs* stated there.

Labels displayed under ASSESS, PLOT, SOLVER, TIMER, TVM, and ∫fdx are printed in the *IOI* but may be not contained in the catalog of functions (CATALOG'FCNS) – some of them denote *system variables*.

If you find some functions in the *IOI* below being unknown to you, simply do not bother: you can perfectly enjoy your *WP43* without those. In particular, many of the operations collected in X.FN may well make sense to mathematicians only.

The *IOI* lists the *name* of each and every *item* provided, be it a command, a constant, a (*sub-*) *menu*, a predefined *label*, an *RoV*, or a *system flag*. These are over 1000 *names*. Just single *characters* and digits are not listed below since these do not interfere with other *items*.

The *IOI* will also support you inventing new *names* for the *variables* and *menus* you create (avoiding name conflicts).

Each user-created *name* of a *variable* or *menu* may consist of up to 7 *characters* and **shall be unique** (i.e. it must not match any other *item name* provided [7] or created before). **Each new *name* shall start with a letter and must not contain a +, −, ×, /, ^, (,), =, !, |, comma, colon, semicolon, or a space.** New program *labels*, on the other hand, do not have to be unique; though multiple use of identical *global labels* is not recommended since it may carry unexpected consequences.

Names are case sensitive (e.g., **M** ≠ **m**) but level insensitive (e.g., **Bn*** equals $\mathbf{B_n}^*$, allowing for easy input, e.g., in equations or catalog searches while the latter notation may ease reading). Thus, feel free to create a variable **Yx** but neither **yx** nor $\mathbf{y_x}$, for instance, since the latter two are in conflict with $\mathbf{y^x}$.

Constants taken from CONST are headed by **#** in programs, so you may use *variables* like **a**, **c**, **e**, **h**, **i**, and **k** there while you must not use these in equations.

Please ensure you have checked these constraints before complaining.

[7] Except single *characters*, cf. above.

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
°C→°F °F→°C	U→ °C → °F etc.	[3] (1) {2}; {1} → {2} Convert temperatures. See pp. 150ff.
0.1s	⏱ 0.1s	[3] (0) Toggles precision of the timer display (1 s / 0.1 s).
10^x	10^x	[2] (1) {1, 2, 3, 8*, 9*, 10} Returns 10^x, the inverse of $\lg(x)$.
1COMPL	BIT ▼ 1COMPL INTS ▲ 1COMPL	[4] (0) Sets *1's complement* mode for operations on *short integers*. See *OMS 2*.
$1/x$	$1/x$	[1] (1) {2, 3, 5, 8*, 9*}; {1} → {2} Inverts the number or matrix x.
2COMPL	BIT ▼ 2COMPL INTS ▲ 2COMPL	[4] (0) Sets *2's complement* mode for operations on *short integers*. See *OMS 2*.
2^x	EXP 2^x	[2] (1) {1, 2, 3, 8*, 9*, 10} Returns 2^x, the inverse of $\mathrm{lb}(x)$.
$\sqrt[3]{x}$	EXP $\sqrt[3]{x}$	[3] (1) {1, 2, 3, 8*, 9*, 10}; ({1} → {2}) Returns the cube root of x.
A	Reserved *variable* for *register* A.	
a a_0		Constants (see pp. 113f).
ABS	CAT. FCNS ABS	Points to |x| on p. 108. Just for backward compatibility.
ACC	ADV ∫fdx ACC EQN ∫f ACC	Reserved *real variable* for the relative accuracy of integration (see *OMS 4*).
ACOS	CAT. FCNS ACOS	Points to ARCCOS. For backward compatibility only.
ActUSB	USB (see p. B-32)	[2] (0) Activates the *FD* (a.k.a. *USB* disk) for data exchange with your *PC*. Returning from it, *WP43* will continue. See *OMS 3*.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
ac.→ha ac.→m² ac$_{US}$→ha ac$_{US}$→m²	U→ A: ha ← acre etc.	[6] (1) {2}; {1} → {2} Convert areas. See pp. 150ff.
ADD	Σ+	[3] (0) Adds the present timer value to STATS.
ADM		Reserved *integer variable* for the *ADM*: 0: DEG, 1: D.MS, 2: RAD, 3: MULπ, 4: GRAD, 5: MIL.
ADV	ADV	*Menu*. See p. 140.
AGM	X.FN AGM	[3] (2) {2, 3}; {1} → {2} Returns the *arithmetic-geometric mean* of x and y. Will throw an error for negative x or y. See p. 316.
AGRAPH	P.FN P.FN2 AGRAPH *s*	[5+] (0) Alpha graphics. Displays a graphics pattern. The *short integer* in the source s specifies a 64-dot column pattern; IP(x) and IP(y) tell the location of the bottom pixel of this column (valid inputs are $0 \le x \le 399$, $0 \le y \le 175$, pixel (0; 0) is bottom left of the *LCD*). So one 64 px high graphic row starting in column 0 requires up to 400 such *integers* to specify – the more blank space is found in this row the less *integers* are needed for describing it.[8] AGRAPH increments x. See also GRAMOD on p. 45.
ALL	DISP ALL *n*	[3+] (0) Sets the display format to show all digits of a *real* or *complex number* as long as displayable (without trailing decimal zeros). For $x \ge 10^{16}$ (or earlier for *complex numbers*), display will switch to SCI (for ~~ALLENG~~) or ENG with maximum displayable precision, using the large font; the same will happen a) for $n = 0$ and $x < 10^{-15}$ or b) for $n \ne 0$ and $x < 10^{-n}$. The limits differ in RBR – see p. 58.

[8] Cf. *HP-42S OM*, pp. 135 - 140, and *HP-42S PET*, pp. 195 - 223.

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
		ALL **0** and ~~ALLENG~~ are *startup default*.[9] See also ENG, FIX, SCI, and examples in *OMS 2.*
ALLENG	SF A	*System flags* (see pp. 127ff)
ALPHA	α	
ALP.IN		
a_{Moon}		Constant (see pp. 113f).
AND	BIT AND	[3] (2) {10} Returns the logical AND of the corresponding bits in x and y (see *OMS 2*).
		(2) {1, 2} → {1} Here, x and y are interpreted before executing AND: zero is 'false'; every nu*mber* ≠ 0 is 'true'.[10]
(ANGLE)		See ∡ on p. 110.
ANGLES	CATALOG VARS ANGLES	*Submenu* of tagged angular *variables* defined at execution time. See pp. 133f.
arccos	TRI arccos	[2] (1) {3, 8*, 9*}; {1, 2} → {4} Returns the tagged *angle* $arccos(x)$.[11]
arcosh	EXP arcosh	[3] (1) {2, 3, 8*, 9*} Returns $arcosh(x)$.
	TRI arcosh	
arcsin	TRI arcsin	[2] (1) {3, 8*, 9*}; {1, 2} → {4} Returns the tagged *angle* $arcsin(x)$.[12]

[9] ALL **0** here works almost like ALL on *HP-42S*.

[10] Input interpretation as on *HP-28S*.

[11] Precisely, it returns the principal value of $arccos(x)$, i.e. a *real* part ∈ [0, π] in ∡r, ∈ [0, 1] in ∡π, ∈ [0°, 180°] in ∡° or ∡", ∈ [0⁻, 1600⁻] in ∡-, or ∈ [0ᵍ, 200ᵍ] in ∡ᵍ. Cf. *ISO/IEC 9899*.

[12] Precisely, it returns the principal value of $arcsin(x)$, i.e. a *real* part ∈ [−π/2, π/2] in ∡r, ∈ [−0.5, 0.5] in ∡π, ∈ [−90°, 90°] in ∡° or ∡", ∈ [−800⁻, 800⁻] in ∡-, or ∈ [−100ᵍ, 100ᵍ] in ∡ᵍ. Cf. *ISO/IEC 9899*.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
arctan	TRI arctan	[2] (1) {3, 8*, 9*}; {1, 2} → {4} Returns the tagged *angle* $arctan(x)$.[13]
arsinh	EXP arsinh TRI arsinh	[3] (1) {2, 3, 8*, 9*} Returns $arsinh(x)$.
artanh	EXP artanh TRI artanh	[3] (1) {2, 3, 8*, 9*} Returns $artanh(x)$.
ASIN	CAT. FCNS ASIN	Points to ARCSIN. For backward compatibility only.
ASLIFT		*System flag* (see pp. 127ff)
ASR	BIT ▲ ASR ***n***	[5+] (1) {10} Shifts all bits in x right n times. Corresponds to a signed division of x by 2^n (with $n \leq 63$).[14] ASR **0** executes as NOP but loads **L**. See SR on p. 87 and *OMS 2*.
ASSESS	FIT ■ ASSESS	[4] (0) Opens a new screen, plots the data points collected in STATS together with the regression curve fitted, and writes its parameters and r^2 next to the plot. See pp. 292ff and *OMS 2*. If BESTF was used to select >1 fit models (see p. 25) then all allowed and calculable models can be displayed sequentially within ASSESS: Use NXTFIT or ▲ to plot the next model, ▼ to plot previous model, and ZOOM for zooming out. Press EXIT to leave the ASSESS screen.
ASSIGN	ASN ***item, location***	[2++] (0) Assigns an *item* (like a function, *menu*, program *label*, or *character*) to a specific target location on the keyboard or in a *user menu*. Your assignments will become effective in *user mode*. Note ASSIGN works in *PEM* and *user mode* as well. See *OMS 6*.

[13] Precisely, it returns the principal value of $arctan(x)$, i.e. a *real* part ∈ [−π/2, π/2] in $\angle^r$, for example (cf. ARCSIN), if SPCRES is set. Else the result interval for ARCTAN becomes, e.g., (−π/2, π/2) in $\angle^r$. Cf. *ISO/IEC 9899*.

[14] Works like n consecutive executions of ASR on *HP-16C*.

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
ATAN	CAT. FCNS ATAN	Points to ARCTAN. For backward compatibility only.
atan2	TRI atan2	[3] (2) {3}; {1, 2} → {4} Returns $arctan2(x,y)$ – see p. 316. Watch the order of parameters. **Example:** **1** ENTER↑ +/- TRI atan2 returns 135°.
atm→Pa	U→ F&p: ▼ Pa ← atm	[6,4] (1) {2}; {1} → {2} Convert pressures and distances. See pp. 150ff.
au→m	U→ x: m ← au	
AUTOFF		*System flags* (see pp. 127ff)
AUTXEQ		
AVIEW	P.FN AVIEW ***s***	[4+] (0) Displays ***s*** left-adjusted in the top numeric row (and below if it needs more space) until the next key is pressed (in programs until the next AVIEW or VIEW occurs). The source address will not be shown (compare VIEW and PROMPT).
A:	U→ A:	*Submenu* for conversions of areas. See p. 150.
$a_\oplus$		Constant (see pp. 113f).
B	Reserved *variable*	for *register* **B**.
BACK	P.FN P.FN2 BACK ***n***	[5+] (0) Jumps ***n*** steps backwards ($0 \le n \le 255$) in a program. E.g., BACK **1** goes to the previous program step. If BACK attempts to cross an END, an error will be thrown. Reaching step 0000 stops program execution. See also SKIP on p. 84.[15]
bar→Pa	U→ F&p: Pa ← bar	[4] (1) {2}; {1} → {2} Converts pressures. See pp. 150ff.

[15] **ATTENTION:** If you edit a section of your routine crossed by one or more BACK, CASE, or SKIP jumps, this may well result in a need to **manually maintain all those statements** individually.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
BATT?	INFO BATT?	[4] (−1) { } → {2} Measures the battery voltage ***U*** between 1.9 V and 3.4 V. Returns ***U*** with a resolution of 1 mV. ***U*** < 2.5 V is considered low, ***U*** < 2.0 V may turn off your *WP43* automatically – a timer running in background will continue! Note the value returned by BATT? is an almost open-circuit voltage (see *App. G*)..
bbl→m³	U→ V: m³ ← barrel	[6] (1) {2}; {1} → {2} Converts volumes. See pp. 150ff.
BC?	BIT BC? ***n***	[4+] (−1) {10} Tests if the specified bit in ***x*** is clear.
BEEP	I/O BEEP	[4] (0) Sounds a sequence of four tones. See also TONE.
BeginP	FIN TVM Begin	[5] (0) Sets “Begin” mode in *TVM*: then payments occur at the begin of each period. Typical for saving plans and leasing. Clears ENDPMT. Compare ENDP.
BestF	FIT ▼ BestF ***n***	[5+] (0) Instructs your *WP43* to select the ‘best’ curve fit model for the accumulated statistical data (i.e. the one returning the maximum *coefficient of determination* (r^2) out of the models allowed).[16] See CORR. Relevant for L.R., $\hat{x}$, and $\hat{y}$. You can accelerate computation significantly by excluding curve fit models making no sense for your data (e.g., for physical or technical reasons). This information is carried by ***n***. Each fit model corresponds to a number as listed: LINF 1 EXPF 2 LOGF 4 POWERF 8 ROOTF 16 HYPF 32 PARABF 64 CAUCHF 128 GAUSSF 256

[16] Works almost like BEST on *HP-42S*.

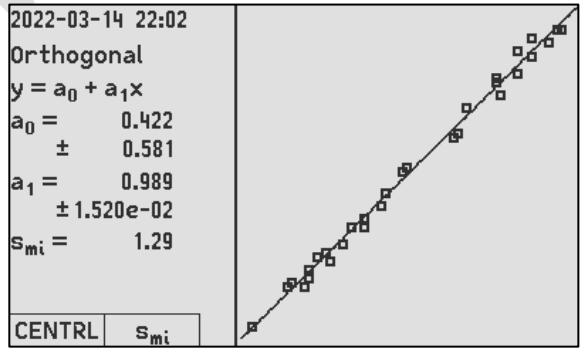
DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
		Take the numbers of all models you can exclude and add them up – the result is ***n***. **Example:** Excluding the three 3-parameter models leads to $n = 64 + 128 + 256 = 448$. Hence, if you are looking for the best-fitting 2-parameter model, call BESTF **448**. We recommend using ASSESS for checking the fit(s). See pp. 292ff.
BestF?	INFO ▼ BestF?	[4] (−1) { } → {1} Returns the current setting of BESTF, encoded as a *long integer* (as explained for BESTF).
$Binom_p$ $Binom_{\blacktriangle}$ $Binom_{\blacktriangle}$ $Binom^{-1}$	PROB Binom: $Binom_p$ etc.	[5] (1) {2}; {1} → {2} *Binomial distribution* with the number of successes ***g*** in **X**, the probability of a success p_0 in **I**, and the sample size ***n*** in **J**. See p. 269. $Binom^{-1}$ returns the maximum number of successes ***m*** for a given probability ***p*** in **X**, p_0 in **I** and ***n*** in **J**.
Binom:	PROB Binom:	*Submenu*. See p. 144.
BIT	BIT	*Menu*. See p. 136.
B_n	X.FN B_n	[3] (1) {1, 2} B_n and B_n^* return the Bernoulli number for an integer $n > 0$ given in **X**, working with different definitions (see p. 267).
B_n^*	X.FN B_n^*	
BS?	BIT BS? ***n***	[4+] (0) {10} Tests if the specified bit in ***x*** is set.
Btu→J	U→ E: J ← Btu	[4] (1) {2}; {1} → {2} Converts energies. See pp. 150ff.
BUZZ	I/O BUZZ	[4] (0) Plays a tone with a frequency ***y*** (in Hz) and duration ***x*** (in ms).
C	Reserved *variable* for *register* **C**.	

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
c c_1 c_2		Constants (see pp. 113f).
cal→J	U→ E: J ← cal	[4] (1) {2}; {1} → {2} Converts energies. See pp. 150ff.
CALC	EQN Solver Calc	[4] (1) Computes the value of the current equation at position x. See *OMS 4*. **ATTENTION:** You shall store x in the corresponding *variable* using the respective *softkey* before calling CALC.
CARRY		*System flag* (see pp. 127ff). Set by SF C.
CASE	P.FN P.FN2 CASE s	[5+] (0) Works like SKIP (see p. 84) but takes the number of steps to skip ($0 \le s \le 255$) from **s**. **Example:** Assume a program section: … 100 CASE 12 101 GTO 01 102 GTO 02 103 GTO 07 104 GTO 05 105 LBL 01 … 132 LBL 02 … 153 LBL 05 … 234 LBL 07 … Executing this program, *r12* will be checked in step 100: if *r12* ≤ 1 then the program will proceed to step 101 and continue with a jump to step 105, for *r12* = 2 the program will go to step 102, etc., resulting in a nice controlled … … dispatcher for $1 \le r12 \le 4$. **ATTENTION:** CASE might surprise you for ***r12*** > 4 here. Take care of the input you provide! And watch f. 15 on p. 24

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
CATALOG	CATALOG	*Catalog* of everything. See pp. 133ff.
CauchF	FIT ▼ CauchF	[5] (0) Selects the *Cauchy* (a.k.a. *Lorentz* or *Breit-Wigner*) peak fit model for L.R. Relevant also for CORR, x̂, and ŷ. See pp. 278ff.
Cauch_p $\text{Cauch}_{\triangle}$ $\text{Cauch}_{\triangle}$ Cauch^{-1}	PROB Cauch: Cauch_p etc.	[5] (1) {2}; {1} → {2} *Cauchy-Lorentz* (a.k.a. *Lorentz* or *Breit-Wigner*) *distribution* with the peak location x_0 specified in **I** and the shape γ in **J**. See p. 272. Cauch^{-1} returns x for a given probability p in **X**, with x_0 in **I** and γ in **J**.
Cauch:	PROB Cauch:	*Submenu*. See p. 144.
CB	BIT CB *n*	[4+] (1) {10} Clears the specified bit in x.
ceil	INTS ceil	[4] (1) {8*}; {1, 2} → {1} Returns the smallest *integer* ≥ x. Compare FLOOR.
CENTRL	FIT PLOT CENTRL	[5] (0) Fits an *orthogonal regression* center line through the data points plotted by PLOT (see p. 69) for an *MSA* (see pp. 295ff); displays also the two fit parameters (y-intercept and slope) with their errors and (for ≥30 data points) the measured precision of the measuring instrument. See s_{mi} on p. 85, s(a) on p. 89, and examples in *OMS 2*. 

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
CF	CF ***n***	[2+] (0) Clears the *flag* specified, i.e. sets it to 0.
	CLR CF ***n***	
	FLAG CF ***n***	
	MODE CF ***n***	
CHARS	CAT. CHARS	*Submenu* of *characters*. See pp. 133ff.
chĭ→m	U→ x: ▲ m ← chĭ	[6] (1) {2}; {1} → {2} Converts distances. See pp. 150ff.
CLall	CLR CLall	[4] (0) Clears all *RoV's*, numbered *user flags*, user assignments, and programs in *RAM*. Modes will stay as they are. See also CLCVAR, CLFALL, CLMALL, CLPALL, CLVALL, CLREGS, CLSTK, RESET, and *OMS 1.*
CLBKUP	CLR CLBKUP	[4] (0) Deletes the backup file created by SAVE.
CLCVAR	CLR CLCVAR	[3] (0) Clears all *variables* used in *current program*: creates all such *variables* not defined yet, sets all *real, complex*, and *integer variables* to zero, all elements of all matrix *variables* to **0.**, all *time variables* to **0:00:00**, all *date variables* to **January 1st** of year **0**, and all *text strings* to zero length.
CLFall	FLAG CLFall	[4] (0) Clears all numbered global and local *user flags* (in programs without confirmation; cf. CF). Lettered *flags* are not cleared since they may coincide with *system flags.*
	CLR CLFall	
CLK	CLK	*Menu*. See p. 133.
CLKEYS	CLR ▲ CLKEYS	[4] (0) Clears all key assignments (cf. ASSIGN).
CLLCD	CLR CLLCD	[4] (0) Clears all pixels ≥ IP(***x***) and ≥ IP(***y***) on the *LCD*. Pixel (0; 0) is bottom left.
CLMall	CLR CLMall	[4] (0) Clears all user defined *menus* in *RAM*.
CLMENU	CLR ▲ CLMENU	[4] (0) Clears all *menu* key definitions for the programmable *menu*. See MENU on p. 59.
	P.FN P.FN2 …	
CLMy...	CLR CLMy...	*Submenu*. See p. 141.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
CLMyM CLMyP CLMyα	CLR CLMy... CLMyM etc.	[4] (0) Clears all user assignments in MyMENU, MyPFN, or Myα.
CLP	CLR CLP	[3] (0) Clears the *current program*. Freed memory is returned to the pool of free space.
CLPall	CLR CLPall	[4] (0) Clears all programs in *RAM*. Cf. CLP.
CLR	CLR	*Menu*. See p. 141.
CLREGS	CLR CLREGS (needs confirmation in *RUM*)	[4] (0) Clears all global and local *GP registers* allocated, i.e. sets them to zero. The contents of the *stack* and **L** are kept; compare CLSTK.
CLSTK	CLR CLSTK 0 FILL	[3] Clears all *stack registers* currently allocated (either **X** … **T** or **X** … **D**). All other *register* contents are kept. Cf. CLREGS.
CLVall	CLR CLVall	[4] (0) Clears all user defined *variables* in *RAM*.
CLX	CLR CLX ← for closed x	[1] (1) Clears *stack register* **X**, disabling *ASL*. Cf. CLREGS.
CLΣ	CLR CLΣ FIT CLΣ STAT CLΣ	[3] (0) Clears STATS and HISTO and releases the memory allocated for them and for the statistical sums. Call CLΣ before accumulating data for a new statistical analysis.
cn(u,m)	X.FN Ellipt cn	[4] (2) {2, 3} Returns the *elliptic cosine* of the argument $u = x$ and parameter $m = y$. See p. 317.
CNST	P.FN CNST *n*	[4+] (−1) { } → {2} Returns the constant stored at position n in the *catalog* CONST (see pp. 113ff). Allows for addressing constants indirectly, also beyond ∞. Note the conversion factors of U→ follow ∞. From $n = 142$ on, auxiliary constants are stored. From $n = 238$ on, kitchen volume conversion factors are found; the last such constant is CNST(266). A total of 368 constants can be addressed.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
COMB	PROB C_{yx}	[3] (2) {1} Returns the number of possible subsets of x objects taken out of a set of y objects (i.e. choose x out of y). No object occurs more than once in a subset, and different orders of the same x objects are not counted separately. Compare PERM.
		(2) {2, 3} See pp. 267f for the formula.
conj	CPX conj	[3] (1) {3, 9*} Returns the *complex conjugate* of x.
CONFIG	CATALOG VARS CONFIG	*Submenu* of all *variables* storing *configurations*. See pp. 133f.
CONST	CONST	*Catalog* of physical, mathematical, astronomical, and surveying constants. Cf. CNST and see pp. 113ff.
CONVG?	TEST CONVG? s	[4+] (0) {2} Checks for convergence by comparing x and y as determined by the lowest five bits of s. a) The very lowest 2 bits set the tolerance limit: 0 = 10^{-14}, 1 = 10^{-24}, 2 = 10^{-32}. b) The next two bits determine the comparison mode using the tolerance limit set: 0 = compare the numbers x and y relatively, 1 = compare them absolutely. c) The top bit tells how special numbers will be treated: 0 = NaN and ±∞ are considered converged, 1 = they are not considered converged. At the bottom line, $s = a + 4b + 16c$.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
CORR	FIT r	[3] (−1) { } → {2} Returns the *coefficient of correlation* r for the current statistical data and fit model. Note r^2 is the *coefficient of determination*. See pp. 278ff.
cos	TRI cos	[2] (1) {2, 3, 8*, 9*}; {1, 4} → {2} Returns the cosine of the angle x (takes x as tagged, else assumes the current *ADM*; see *OMS 2*).
cosh	EXP cosh TRI cosh	[3] (1) {2, 3, 8*, 9*} Returns the hyperbolic cosine of x.
COV	FIT cov	[3] (−1) { } → {2} Returns the *population covariance* for the two data sets {x} and {y} entered via Σ+. See s_{XY} for the *sample covariance* and pp. 278ff.
$cup_I \rightarrow ml$	U→ V: ▲ ▲ ml ← cup_I	[7,6] (1) {2}; {1} → {2} Convert volumes. See pp. 150ff.
$cup_{UK} \rightarrow ml$	U→ V: ▲ ml ← cup_{UK}	
$cup_{US} \rightarrow ml$	U→ V: ▼ ml ← cup_{US}	
CPX	CPX	*Menu*. See p. 141.
CPXRES		*System flag* (see pp. 127ff), set by SF I (imag.).
CPXS	CAT. VARS CPXS	*Submenu* of all *complex variables* defined at execution time. See pp. 133f.
CPXSLV	EQN Solver cpxSlv	[4] Finds *complex* (and *real*) roots of the current equation in EQN (see SOLVE, on the other hand, which will find only *real* roots). Two initial guesses of the root may be supplied in **X** and **Y** when calling CPXSLV. Its results follow the rules of SOLVE (see p. 86). There is no way to force CPXSLV to find a root with imaginary part ≠ 0 (or = 0) – you should try with different (or swapped) first guesses. With a root found, press CALC to verify.
CPX?	TEST ▲ CPX?	[4] (0) Checks if x is *complex*. Returns true if **X** contains data of type 3 or 9 with nonzero *imaginary* part (see also *App. B*).

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
cross	MATX cross	[4] (2) {8} Requires two *real* 3D vectors x and y and returns their cross product $y \times x$. Crossing *2D* vectors works as it does for *complex numbers*.
	CPX cross	[3] (2) {3} → {2} When two *complex numbers* are crossed, your *WP43* simply returns a *real number* equal to the signed *magnitude* of the resulting moment vector. See an example in *OMS 2*.
crt→g	U→ m: ▼ g ← carat	[5,6,4] (1) {2}; {1} → {2} Convert masses and distances. See pp. 150ff.
crt→kg	U→ m: ▲ kg ← carat	
cùn→m	U→ x: ▲ m ← cùn	
cwt→kg	U→ m: kg ← cwt	
CX→RE	CPX CX→RE	[1] (−1) {3} → {2}; {9} → {8} Cuts a closed *complex* object x, putting either • (for ⌞) its *real* part in Y and its *imaginary* part in X or • (for ⊙) its *magnitude* in Y and its *phase* in X.
	CC (working in *RUM* only)	
D	Reserved *variable*	for *register* D.
DATE	CLK DATE	[3] (−1) { } → {6} Recalls date and time from the real-time clock and displays it in the format selected by you (see D.MY, M.DY, and Y.MD). Furthermore, DATE shows the day of week (see *OMS 2*).
DATES	CATALOG VARS DATES	*Submenu* of all *date variables* defined at execution time. See pp. 133f.
DATE→	CLK DATE→	[3] (−2) {2, 6} → {1} Takes x as a *date* in the format selected (or a *real number* in corresponding format) and pushes its three components as *integers* on the *stack*. Reversible by →DATE (see p. 106).

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
DAY	CLK DAY	[4] (1) {2, 6} → {1} Takes ***x*** as a *date* in the format selected (or a *real number* in corresponding format) and extracts the day. See also WDAY on p. 94.
day→s	U→ Time: s ← day	[4] (1) {2}; {1} → {2} Converts time. See pp. 150ff.
DBLR	INTS DBLR	[4] {10} Double *word* length commands for remainder, multiplication, and division. DBL× takes ***x*** and ***y*** as factors as usual but returns the product in **X** and **Y** (with its most significant bits in **X**). This result may be up to 128 bits long. DBL/ and DBLR accept a double size dividend in **Y** and **Z** (with its most significant bits in **Y**), the divisor in **X** as usual, and return the result in **X**. Thus, DBL/ and DBLR are *triadic* functions.
DBL×	INTS DBL×	
DBL/	INTS DBL/	
dB→fr	U→ ▲ dB→field ratio	[4] (1) {2}; {1} → {2} Convert ratios. See pp. 150ff.
dB→pr	... dB→power ratio	
DEC	LOOP DEC ***r***	[4+] (0) {1, 2, 10} Decrements ***r*** by 1. Does not load **L** even for target address **X**.
DECIM.		*System flag* (see pp. 127ff)
DECOMP	PART DECOMP	[3] (−1) {1, 2} → {1} Decomposes ***x*** (after converting it to an *improper fraction*, if applicable), returning a *stack* [*denominator(x), numerator(x),* …]. This works with maximum precision always, regardless of the settings of D.MAX, DENFIX, and DENANY. Can be reverted by division. **Example:** For ***x*** = **2.25**, DECOMP returns ***y*** = **9** and ***x*** = **4**.
DEG	MODE DEG	[3] (0) Sets the *ADM* to *decimal degrees*.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
DEG→	[↙→] DEG→	[4] (1) {1, 2, 4} → {4} Converts *angles* as described on p. 161.
DelITM	CLR [▲] DelITM	[4] (0) Deletes a user-defined *item* from memory.
DelMall	CLR DelMall	[4] (0) Deletes all user-defined *menus* or *variables* from memory. Cf. CLMall and CLVall.
DelVall	CLR DelVall	
DENANY		*System flags* (see pp. 127ff)
DENFIX		
DENMAX	MODE DENMAX	[4] (1) Specifies the maximum denominator for fractions ($1 \le x \le 9999$). For $x < 1$ or $x > 9999$, D.MAX will be set to 9999.[17] For $x = 1$, the current D.MAX setting is recalled, replacing x.
DET	CAT. FCNS DET	Equals |M| on p. 108. For backward compatibility only.
DISK?	INFO DISK?	[4] (0) Returns *SM*'s *DMCP* information about the *FD* looking like this → Press any key to close. Disk Info Disk State: READY Storage: OK Storage info: Disk block count: 1536 Bad block count: 0 1st good block: 0 Note for information about free space on the *FD* you have to connect it to a *PC* (e.g., by ActUSB) and call disk *Properties* there.
DISP	DISP	*Menu*. See p. 141.
DMCP	MODE DMCP	[4] (0) Returns to *SM*'s *DMCP menu.* See p. B-32 and *App F*.
DMY		*System flag* (see pp. 127ff)
dn(u,m)	X.FN Ellipt dn	[4] (2) {2, 3} Returns the elliptic *delta amplitudinis* of the argument $u = x$ and parameter $m = y$. See p. 317.

See *OMS* 6.

[17] Works like /c on the *HP-35S*, but with a greater maximum denominator.

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
dot	MATX **dot**	[4] (2) {8} → {2}; {9} → {3} Requires two matrices in **X** and **Y** and returns their dot (scalar) product. This product is the sum of the products of the corresponding elements in both matrices. Note both matrices must match in size, else DOT will throw an error. See *OMS 2.*
	CPX **dot**	[3] (2) {3} → {2} Returns $Re(\boldsymbol{x})\ Re(\boldsymbol{y}) + Im(\boldsymbol{x})\ Im(\boldsymbol{y})$
DRAW	EQN **Solver** **Draw**	[4] (0) Opens a new screen and draws the graph for the current equation within the boundaries specified. Cf. *OMS 4* for examples.
	EQN **∫f** **Draw**	[4] (0) Opens a new screen and draws the integrand within the limits specified (**↓Lim** and **↑Lim**).
		The drawing is based on 41 calculated points.
DROP	DROP↓	[2] Drops $\boldsymbol{x}$ … from the *stack*. See *OMS 1.*
DROPy	STK **DROPy**	[3] Drops $\boldsymbol{y}$
DSE	LOOP **DSE** ***r***	[3+] (0) {1, 2, 10} Most useful in a program running. Given ***r*** = `ccccc.fffii`, DSE decrements ***r*** by `ii` and will skip next program step if `ccccc` ≤ `fff` then (cf. the rear side of your *WP43*). If ***r*** features no fractional part then `fff` is 0. If `ii` = 0, `ccccc` will be decremented by 1. DSE does not load **L** even for target address **X**. Note that neither `fff` nor `ii` can be negative, and DSE makes sense with `ccccc` > 0 only. In *RUM*, DSE returns **true** for `ccccc` > `fff` and **false** for `ccccc` ≤ `fff`.
DSL	LOOP **DSL** ***r***	[3+] (0) {1, 2, 10} Works like DSE but skips if `ccccc` < `fff`.
DSP	DSP ***n***	[2+] (0) Specifies ***n*** quickly for ALL, ENG, FIX, or SCI, whatever is currently chosen.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
DSTACK	DISP ▲ DSTACK ***n***	[5+] (0) Sets the number of *stack registers* to be displayed. For an input of 1, only x will be shown directly above the *menu section*; for 2, x and y will be displayed; maximum input is 4. Expanded views of, e.g., matrices and dual- or multi-level returns like SUM will work as described regardless of the DSTACK set. Command input will be echoed directly below … … the *status bar* and also error messages will be displayed there always. This command is dedicated to old-school *RPN* users who may feel distracted by a multitude of *stack registers* displayed changing simultaneously while only the lowest are really relevant.
DSZ	LOOP DSZ ***r***	[3+] (0) {1, 2, 10} Decrements r by 1 and skips the next step if $-1 < r < +1$ thereafter. Does not load $\mathbf{L}$ even for target address $\mathbf{X}$.[18]
DUMP	I/O DUMP	[3] (0) Dumps x (in full precision) onto the *FD* in `SAVFILES/reg`***x*-yyyymmdd-hhmmssdd**`.tsv`. This is how to export also very long integers (>298 digits) from the calculator. See *App. E*.
D.MAX		Reserved *integer variable* for the maximum denominator in fractions, set by DENMAX.
D.MS	d.ms for closed x	[2] (0) Sets the *ADM* to *sexagesimal degrees*.
D.MS→	↙→ D.MS→	[4] (1) {1, 2, 4} → {4} Convert *angles* as described on p. 161.
D.MS→D	↙→ D.MS→D	
D.MY	CLK ▲ D.MY	[4] (0) Sets the *date* format `dd.mm.yyyy`.
D→D.MS	↙→ D→D.MS	[4] (1) {1, 2, 4} → {4} Converts *angles* as described on pp. 161f.

[18] DSZ is known from *HP-65*, *HP-19C/29C*, *HP-67*, and *HP-16C*.

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
D→J	CLK D→J	[4] (1) {2, 6} → {1} Takes ***x*** as a *date* in the format selected (or a *real number* in corresponding format) and converts it to a *Julian day number* according to the J/G setting (see p. 50).
D→MIL	↙→ ▲ D→MIL	[4] (1) {1, 2, 4} → {4} Convert *angles* as described on pp. 161f.
D→R	↙→ D→R	
e		Constant (see pp. 113ff).
EDIT	EDIT (see p. B-32)	[2] (0) In *RUM*, opens the object ***x*** for editing. In *PEM*, opens the current program step for editing (the step the *PP* is pointing to).
EIGVAL	MATX ▲ EIGVAL	[5] (−1) {8, 9} Evaluates the matrix ***x*** and pushes the matrix of its eigenvalues on the *stack*.
EIGVEC	MATX ▲ EIGVEC	[5] (−1) {8, 9} Evaluates the matrix ***x*** and pushes the matrix of its eigenvectors (as column vectors) on the *stack*.
Ellipt	X.FN Ellipt	*Submenu* for elliptic functions. See p. 146.
END	P.FN END	[3] (0) Last command in a program and terminal for searching *local labels* (see *OMS 3*). Can't be skipped by a test. Works like RTN in all other aspects (see p. 79).
EndP	FIN TVM End	[5] (0) Sets "End" mode in *TVM*: Payments occur at the end of each period. Typical for loans and investments. Sets ENDPMT. Cf. BEGINP.
ENDPMT		*System flag* (see pp. 127ff)
ENG	DIS.P ENG ***n***	[3+] (0) Sets the engineer's display format (see *OMS 2*) with $0 \le n \le 15$. E.g., 12345.67 will display as 12.35×10^{3} for ENG **3** and as 12.3457×10^{3} for ENG **5**. A factor 10^{0} will not be displayed.

L will not be loaded. See *OMS 2*.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
ENORM	MATX ENORM	[4] (1) {8, 9} → {2} Calculates the *Euclidean* norm of the matrix in **X**, defined as the square root of the sum of squares of all matrix elements. For a vector, ENORM returns its length. Compare |x| on p. 108.
ENTER↑	ENTER↑	[1] Separates two inputs. Closes x and copies it into **Y**. Disables *ASL*, so **X** can take a new entry without interfering with the prior one. See p. 167, *App. H*, and *OMS 1*. Note that ENTER↑ ↶ works like EXIT.
ENTRY?	TEST ENTRY?	[4] (0) Checks the (internal) entry *flag*. It is set if … • any *character* is entered in *AIM*, or • any command is accepted for entry (be it via ENTER↑, a function key, or R/S with a partial command line). Useful in routines, e.g., after PAUSE.
EQN	EQN	*Menu*. See p. 141.
EQ.DEL	EQN DELETE	[4] (0) Deletes the current equation.
EQ.EDI	EQN EDIT	[3] (0) Opens the *Equation Editor* to edit the current equation or to enter a new one.
EQ.NEW	EQN NEW	
erf	X.FN erf	[3] (1) {2}; {1} → {2} Returns the error function or its complement. See pp. 316ff.
erfc	X.FN erfc	
ERR	P.FN P.FN2 ERR *n*	[5+] (0) Raises the error code specified (see *App. C*). The consequences are the same as if the corresponding error really occurred, so, e.g., a running routine will be stopped and the error message will be thrown. Compare MSG.
EVEN?	TEST EVEN?	[4] (0) Checks if x is *integer* (see p. 48) and even.
(EXIT)	EXIT	See p. 170. Note that EXIT ↶ works like NOP.

See OMS 4

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
EXITall	P.FN P.FN2 EXITall	[4] (0) Exits all *menus* in *RUM*. In *PEM*, enters a step EXITALL.
	EXIT pressed for >1 s	[1] (0) Exits all *menus*.
EXP	EXP	*Menu*. See p. 141.
ExpF	FIT ▼ ExpF	[4] (0) Selects the exponential fit model for L.R. Relevant also for CORR, $\hat{x}$, and $\hat{y}$. See pp. 278ff.
Expon_{p} $\text{Expon}_{\text{▲}}$ $\text{Expon}_{\text{▲}}$ Expon^{-1}	PROB Expon: Expon_{p} etc.	[5] (1) {2}; {1} → {2} *Exponential distribution* with the rate λ in $\mathbf{I}$ (see pp. 269ff). Expon^{-1} returns the survival time t_s for a given probability p in $\mathbf{X}$, with λ in $\mathbf{I}$.
Expon:	PROB Expon:	*Submenu*. See p. 144.
EXPT	PART EXPT	[3] (1) {1, 2} → {1} Returns the exponent h of $x = m \cdot 10^h$. Compare MANT.
E(m)	X.FN Ellipt E(m)	[4] (1) {2, 3} Returns the *complete elliptic integral of the 2nd kind* for $m = x$. See p. 317.
E(φ,m)	X.FN Ellipt E(φ,m)	[5] (1) {2, 3} Returns the *incomplete elliptic integral of the 2nd kind* for $\varphi = x$ for $m = y$. See p. 317.
E:	U→ E:	*Submenu* for energy conversions. See p. 150.
e		Constant (see pp. 113ff).
e^x	e^x	[1] (1) {2, 3, 8*, 9*, 10}; {1} → {2} Returns the exponential e^x, the inverse of $\ln(x)$. Note $e^{i\pi} = -1$.
e^x-1	EXP e^x-1	[3] (1) {2, 8*} For $x \approx 0$, this returns a more accurate result for the fractional part than e^x does.
F		Constant (see pp. 113ff).

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
FACTOR	X.FN FACTOR	[4] (1) {1, 10} → {8} Factorizes ***x***. Returns a matrix with the prime factors of ***x*** in row 1 and the corresponding exponents in row 2. See p. 268 for an example and more. See pp. 195f for limitations.
FASTFN		*System flag* (see pp. 127ff)
FB	BIT FB ***n***	[4+] (1) {10} Inverts ('flips') the specified bit in ***x***.
FBR	α.FN FBR	[4] (0) Font browser. Shows all *characters* designed for your *WP43*.
FCNS	CATALOG FCNS	*Catalog* of all functions provided. See pp. 133ff.
FC?	FLAG FC? ***n***	[3+] (0) Tests if the specified *flag* is clear.
FC?C FC?F FC?S	FLAG FC?C ***n*** etc.	[4+] (0) Test if the specified *flag* is clear. Will clear, flip, or set this *flag* after testing, respectively.
fēn→m	U→ x: ▼ m ← fēn	[6] (1) {2}; {1} → {2} Converts distances. See pp. 150ff.
FF	FLAG FF ***n***	[3] (0) Flips the *flag* specified.
FIB	X.FN FIB	[4] (1) {1} Returns the Fibonacci number (max. input is 4791, returning a 1001-digit *integer*). (1) {2, 3} Returns the extended Fibonacci number. See pp. 267f.
FILL	FILL	[2] Fills the entire *stack* with ***x***.
FIN	FIN	Menus. See p. 142.
FIT	FIT (see p. B-32)	
FIX	DISP FIX ***n***	[3+] (0) Sets fixed point display format (see *OMS 2*); $0 \le n \le 15$ is the number of decimals of the mantissa to be displayed.
FLAG	FLAG	*Menu*. See p. 142.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
floor	INTS floor	[4] (1) {8*}; {1, 2} → {1} Returns the greatest *integer* ≤ x. Cf. CEIL.
FP	PART FP # F for closed x	[3] (1) {1, 2, 8*, 10} Returns the fractional part of x. Compare IP.
FP?	TEST FP?	[4] (0) Tests x for having a fractional part ≠ 0 .
$F_p(x)$ $F_{\blacktriangle}(x)$ $F_{\blacktriangle}(x)$ $F^{-1}(p)$	PROB F: $F_p(x)$ etc.	[4] (1) {2}; {1} → {2} *Fisher's F distribution*. The *degrees of freedom* shall be specified in **I** and **J**. See pp. 269ff. ($F_{\blacktriangle}(x)$ equals Q(F) on *HP-21S*.)
FRACT FRCSRN		*System flags* (see pp. 127ff)
fr→dB	U→ ▲ db←field ratio	[4] (1) {2}; {1} → {2} Converts ratios. See pp. 150ff.
FS?	FLAG FS? *n*	[3+] (0) Tests if the specified *flag* is set.
FS?C FS?F FS?S	FLAG FS?C *n* etc.	[4+] (0) Test if the specified *flag* is set. Will clear, flip, or set this *flag* after testing, respectively.
ftm.→m	U→ x: ▲ m ← fathm	[6,5] (1) {2}; {1} → {2} Convert distances. See pp. 150ff.
ft.→m	U→ x: m ← foot	
ft_{US}→m	… ▼ m ← surv.foot$_{US}$	
FV	FIN TVM FV	Reserved *real variable* for the future value of your investment or loan in *TVM*. See *OMS 5.*
fz_{UK}→m^3	U→ V: m^3 ← $floz_{UK}$	[6,7] (1) {2}; {1} → {2} Convert volumes. See pp. 150ff.
fz_{UK}→mℓ	… V: ▲ mℓ ← $floz_{UK}$	
fz_{US}→m^3	… V: m^3 ← $floz_{US}$	
fz_{US}→mℓ	… V: ▼ mℓ ← $floz_{US}$	

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
F_α F_δ		Constants (see pp. 113ff).
F(φ,m)	X.FN Ellipt F(φ,m)	[5] (1) {2, 3} Returns the *incomplete elliptic integral of the 1st kind* for $\varphi = x$ for $m = y$. See p. 317.
F:	PROB F:	*Submenu* for the *F distribution*. See p. 144.
f'	EQN f'	*Submenus* for interactively computing the 1st or 2nd derivative of a given equation. See *OMS 4*.
f''	EQN f''	
f' (x)	ADV f' (x) ***label***	[3+,4+] {1, 2} → {2}
f''(x)	ADV f''(x) ***label***	f'(x) [f''(x)] returns the 1st [2nd] derivative of the function $f(x)$ at position x. This $f(x)$ must be specified in a routine starting with LBL ***label***. On return, **Y**, **Z**, and **T** will be cleared and the position x will be in **L**. See *OMS 4*. **ATTENTION:** f'(x) and f''(x) fill all *stack registers* with x before calling the routine specified.
F&p:	U→ F&p:	*Submenu* for force and pressure conversions. See p. 150.
G G_0		Constants (see pp. 113ff).
GAP	DISP ▲ GAP ***n***	[5+] (0) Defines the interval for inserting narrow blank digit group separators in *DT 1*, *2*, *3*, *5*, *8*, and *9* (for *DT 10*, intervals are fixed: 4 digits for binary, 2 for bases 4, 8, and 16, and 3 for all other bases; separators in input will always be inserted like for *reals*). See *OMS 2*. Actually, n = **4** formats the Chinese way and n = **3** for the rest of the world – except South Asia, which needs a more complex format (to be selected by SETIND). For n < **3**, no group separators will be displayed in any number at all.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
$gal_{UK}\rightarrow\ell$	U→ ▼ V: ▲ ℓ ← gal_{UK}	[7,5] (1) {2}; {1} → {2} Convert volumes. See pp. 150ff.
$gal_{US}\rightarrow\ell$	U→ ▼ V: ▼ ℓ ← gal_{US}	
$gal_{UK}\rightarrow m^3$	U→ ▼ V: m^3 ← gal_{UK} etc.	
$gal_{US}\rightarrow m^3$		
GaussF	FIT ▼ GaussF	[5] (0) Selects the *Gauss* peak fit model for L.R. Relevant also for CORR, $\hat{x}$, and $\hat{y}$. See pp. 278ff.
G_C		Constant (see pp. 113ff).
GCD	INTS GCD	[4] (2) {1; 10} Returns the *Greatest Common Divisor* of x and y. This will always be positive. [19]
g_d	X.FN g_d	[4] (1) {2, 3}; {1} → {2} Returns the Gudermannian function or its inverse. See p. 319.
g_d^{-1}	X.FN g_d^{-1}	
g_e		Constant (see pp. 113ff).
$Geom_p$	PROB ▼ Geom: $Geom_p$ etc.	[5] (1) {2}; {1} → {2} *Geometric distribution*: The *CDF* returns the probability for a 1st success after x Bernoulli experiments. The probability p_0 for a success in each such experiment must be supplied in **I**. $Geom^{-1}$ returns the number of failures f before 1st success for given probability p in **X**. See pp. 269ff.
$Geom_{\blacktriangle}$		
$Geom_{\blacktriangle}$		
$Geom^{-1}$		
Geom:	PROB ▼ Geom:	*Submenu*. See p. 144.
$GM_{\oplus}$		Constant (see pp. 113ff).
GRAD	MODE GRAD	[4] (0) Sets the *ADM* to *grades*, a.k.a. *gon*.
GRAD→	∠→ GRAD→	[4] (1) {1, 2, 4} → {4} Converts *angles* as described on pp. 161f.

[19] $\mathrm{GCD}(x;y) = \left| {}^{xy}/_{\mathrm{LCM}(x;y)} \right|$. See also *LCM*.

TN: GCD correspond à *PGCD* en français. / GCD entspricht *ggT* auf Deutsch.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
GRAMOD		Reserved *short integer variable* determining how the AGRAPH pattern (p. 21) is displayed: [20] 0: It is merged (OR-ed) with the existing display (*startup default*). 1: It overwrites all pixels in that part of the *LCD* 2: Duplicate "on" pixels get turned "off". 3: It is XOR-ed with the existing display.
GROW		*System flag* (see pp. 127ff)
GTO	GTO ***label***	[2+] (0) In *PEM*, inserts an unconditional branch to ***label***. Else positions the *PP* to ***label***.
GTO.	GTO . ***nnn***	[3+,4] (0) Puts the *PP* to … *local label* ***n*** (specify ≤ 2 digits).
		step ***nnn*** in *current program* – specify ≥ 3 digits until becoming definite.
	GTO . ***label***	the *global label* specified (shall be picked from PROG).
	GTO . .	the end of used program memory, i.e. right to the final END, where you can start entering a new program.
GTO↑	GTO ▲	[3] (0) Puts the *PP* directly after … previous END, i.e. to the top of *current program* (see *OMS 3*); if being there already, jumps to the top of previous program.
GTO↓	GTO ▼	next END, i.e. to the top of next program.
g→crt	U→ m: ▼ g→carat	[5] (1) {2}; {1} → {2} Convert masses. See pp. 150ff.
g→oz.	… g→oz.	
g→troz	… g→troy oz.	
$g_{\oplus}$		Constant (see pp. 113ff).
h		Constant (see pp. 113ff).

[20] Working like *flags* 34 and 35 in *HP-42S*.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
ha→ac	U→ A: ha → acre	[6] (1) {2}; {1} → {2} Convert areas. See pp. 150ff.
ha→ac$_{US}$	… ha → acre$_{US}$	
ha→m^2	… m^2 ← ha	
HIDE	DISP ▲ HIDE	[5] (0) Hides all *real numbers* or parts with absolute values $<10^{-n}$ with $n = \mathrm{IP}(x)$, $12 \le n \le 99$ (i.e. displays '0.' instead of those). Useful, e.g., in matrices. HIDE **00** resets, so everything allowed by RANGE will be displayed again; see RANGE (p. 72) and pp. 195ff.
HIDE?	DISP ▲ HIDE?	[4] (-1) { } → {1}
	INFO ▲ HIDE?	Returns the current setting of HIDE.
HISTO		Reserved *matrix variable* for histograms.
HISTOG	FIT HISTOG	*Submenu*. See p. 145.
	STAT HISTOG	
HISTOX	… HISTOG HISTOX	[4] (0) Evaluates column 1 (or 2) of STATS and stores the results in HISTO.
HISTOY	… HISTOG HISTOY	
H$_n$	X.FN Orthog H$_n$	[4,5] (2) {2}; {1} → {2}
H$_{np}$	… H$_{np}$	*Hermite polynomials* for probability (H$_n$) and physics (H$_{np}$). See p. 302.
HPLOT	… HISTOG HPLOT	*Submenu*. Cf. HISTOG.
hp$_E$→W	U→ P: W ← hp$_E$ etc.	[4] (1) {2}; {1} → {2} Convert powers. See pp. 150ff.
hp$_M$→W		
hp$_{UK}$→W		
Hyper$_p$	PROB Hyper: Hyper$_p$ etc.	[5] (1) {2}; {1} → {2} *Hypergeometric distribution* with the number of successes ***g*** in **X**. The probability of a success ***p0*** shall be in **I**, the sample size ***n*** in **J**, and the batch size ***n0*** in **K**. See pp. 269ff. Then Hyper^{-1} returns the maximum number of successes ***m*** for a given probability ***p*** in **X**.
Hyper$_{\blacktriangle}$		
Hyper$_{\blacktriangle}$		
Hyper^{-1}		

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
Hyper:	PROB Hyper:	*Submenu*. See p. 144.
HypF	FIT ▼ HypF	[5] (0) Selects the hyperbolic fit model for L.R. Relevant also for CORR, x̂, and ŷ. S. pp. 278ff.
ħ		Constant (see pp. 113ff).
I	Reserved *variable* for *register* **I**.	
IDIV	INTS IDIV	[4] (2) {1, 10}; {2} → {1} Integer division, working like / + IP.[21]
IDIVR	CATALOG FCNS IDIVR	{1, 2, 10} Like IDIV but returns also the remainder in **Y**.[21]
IGN1ER		*System flag* (see pp. 127ff)
iHg→Pa	U→ F&p: Pa ← in.Hg	[5] (1) {2}; {1} → {2} Converts pressures. See pp. 150ff.
Im	CPX Im PART Im	[4] (1) {2, 3} → {2}; {9} → {8} Returns the imaginary part of x. Compare RE.
INC	LOOP INC *r*	[4+] (0) {1, 2, 10} Increments r by 1. Does not load **L** even for target address **X**.
INDEX	MATX INDEX *name*	[4+] (1) Indexes the matrix[22] specified. A matrix shall be indexed before I+, I−, J+, J−, M.GET, M.PUT, M.R⇄R, RCLEL, STOEL, RCLIJ, and STOIJ can be used. You can also index a matrix by editing it (via M.EDIT or M.EDIN); after exiting the *Matrix Editor*, this matrix is no longer indexed.[23]
INFO	INFO	*Menu*. See p. 142.

[21] See *OMS 2* for the *DTs* of quotient and remainder.

[22] Your WP43 allows for matrices in *variables*, but also in *registers*. A named matrix was explicitly required on the *HP-42S* only.

[23] See also *Matrix Utility Functions* in the *HP-42S OM*, pp. 223ff.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
INPUT	P.FN INPUT *r*	[3+] Works in programs only: Recalls the content of the source specified into X, displays the name of the source along with *r*, and halts program execution, allowing you to enter or calculate a value. Pressing R/S then stores *x* into said destination and continues program execution – pressing EXIT instead cancels INPUT, so R/S thereafter will continue with the source content as it was. If you use an input *variable* undefined at execution time, INPUT automatically creates this *variable* with an initial value of zero. **ATTENTION 1:** While single-stepping through a routine, close input with ▼ instead of R/S. **ATTENTION 2:** After INPUT, Y will not contain what was in X before INPUT was called. **ATTENTION 3:** Take care of the *stack* when you calculate while INPUT is open – if your routine uses higher *stack* levels, your calculations may have side effects.
INTING		*System flag* (see pp. 127ff)
INTS	INTS	*Menu*. See p. 142.
INT?	TEST INT?	[4] (0) Tests *x* for being *integer*. Also *reals* with FP(*x*) = 0 are rated as *integer* here. Cf. FP?. Note: E.g., for 5., both INT? and REAL? will return `true`. For 5, only INT? will return `true`.
INVRT	CAT. FCNS INVRT	Equals $[M]^{-1}$ (p. 108). For backward compatibility only.
in.→m	U→ x: m ← inch	[5,6] (1) {2}; {1} → {2} Convert distances. See pp. 150ff.
in.→mm	... ▼ mm ← inch	
IP	PART IP	[3] (1) {1, 8*}; {2, 10} → {1} Returns the integer part of *x*. Cf. FP.
	# | for closed *x*	

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
ISE	LOOP ISE ***r***	[3+] (0) {1, 2, 10} Given `cccccc.fffii` in ***r***, ISE increments ***r*** by `ii`, skipping next program step if `cccccc` ≥ `fff` then (cf. the rear side of your *WP43* and DSE on p. 36). If ***r*** features no fractional part then `fff` is 0. If `ii` = 0, `cccccc` will be incremented by 1. ISE does not load **L** even for target address **X**. Note that neither `fff` nor `ii` can be negative, but `cccccc` can. In *RUM*, ISE returns **true** for `cccccc` < `fff` and **false** for `cccccc` ≥ `fff`.
ISG	LOOP ISG ***r***	[3+] (0) {1, 2, 10} Works like ISE but skips if `cccccc` > `fff`.
ISM		Reserved *integer variable* for the *integer sign mode* of *short integers*: −1: sign and mantissa mode, 0: unsigned, 1: 1's complement, and 2: 2's complement (*SDC*).
ISM?	INFO ISM?	[3] (−1) { } → {1} Returns the *integer sign mode* set for *short integers*: **true** −1 for sign and mantissa mode, **false** 0 for unsigned, **true** 1 for 1's complement, or **true** 2 for 2's complement.
ISZ	LOOP ISZ ***r***	[3+] (0) {1, 2, 10} Increments ***r*** by 1, skipping next program step if $-1 < s < +1$ thereafter. ISZ does not load **L** even for target address **X**.[24]

[24] ISZ is known from *HP-19C/29C*, *HP-67*, and *HP-16C*.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
I_{xyz}	X.FN I_{xyz}	[4] (3) {1, 2} Returns the *regularized Beta function*; see p. 319.
I+	MATX ▲ I+	[4] (0) Increments or decrements the row index ***i*** of the indexed matrix. See INDEX and also J+, J−, RCLEL, STOEL, RCLIJ, and STOIJ.
I-	MATX ▲ I-	
I/O	I/O	*Menu*. See p. 142.
i%/a	FIN TVM i%/a	Reserved *real variable* for the annual interest rate of your investment or loan in *TVM*. See *OMS 5.*
i		Constant (see pp. 113ff).
J	Reserved *variable* for *register* **J**.	
jīn→kg	U→ m: ▲ kg ← jīn	[6] (1) {2}; {1} → {2} Converts masses. See pp. 150ff.
$J_y(x)$	X.FN $J_y(x)$	[4] (2) {2}; {1} → {2} Returns the *Bessel function of first kind* and order ***y***. See p. 319.
J+	MATX ▲ J+	[4] (0) Increments or decrements the column index ***j*** of the indexed matrix. If GROW is set and ***i*** and ***j*** are pointing at the last element of the matrix, executing J+ creates a new row at the end of the matrix. See INDEX and also I+, I−, RCLEL, STOEL, RCLIJ, and STOIJ.
J-	MATX ▲ J-	
J/G	CLK ▲ J/G	[5] (0) {2} Takes ***x*** specifying the date the *Gregorian* calendar became valid in the region you are interested in. This shall be entered as a *real number* reflecting the crucial *Gregorian* date, formatted according to the *date display mode* set (DMY, MDY, or YMD).
J/G?	INFO ▲ J/G?	[4] (-1) { } → {6} Returns the current setting of J/G. *Startup default* is **1752-09-14**.
J→Btu	U→ E: J → Btu etc.	[4] (1) {2}; {1} → {2} Convert energies. See pp. 150ff.
J→cal		

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
J→D	CLK J→D	[4] (1) {1} → {6} Takes ***x*** as a *Julian day number*[25] and converts it to a common *date* according to J/G and the date format selected.
J→Wh	U→ E: J → Wh	[4] (1) {2}; {1} → {2} Converts energies. See pp. 150ff.
K	Reserved *variable*	for *register* K.
k		Constant (see pp. 113ff).
KEY		See KEYG and KEYX.
KEYG	P.FN P.FN2 KEYG ***key#***, ***label***	[4++] Specifies the *label* to be branched to (KEYG) or called (KEYX) when a particular *softkey* is pressed. KEYG and KEYX work in *PEM* only and will be translated to a program step (on) KEY ***key#*** GTO ***label*** or (on) KEY ***key#*** XEQ ***label*** , respectively. Key numbers go from 1 to 21 with 1 corresponding to F1, 9 to F3, and 14 to F2, for example. 19 - 21 represent ▲, ▼, and EXIT. See *OMS 3*.
KEYX	P.FN P.FN2 KEYX ***key#***, ***label***	
KEY?	TEST KEY? ***r***	[4+] (0) Tests if a key was pressed while a routine was running or paused. If no key was pressed in that interval then the next program step after KEY? will be executed; else it will be skipped and the code of said key will be stored in ***r***. Key codes reflect the rows and columns on the keyboard (see *OMS 3*).[26] See PUTK.

[25] Note a *Julian day number* differs from a *Julian date*. See corresponding articles in *Wikipedia* for more information about these counting numbers.

TN: *Julian day number* translates to «jour Julien» in French and to "Julianische Tageszahl" in German.

[26] See also GETKEY on *HP-42S*.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
kg→crt	U→ m: ▲ kg → carat	[5,4,6] (1) {2}; {1} → {2} Convert masses. See pp. 150ff.
kg→cwt	… kg → cwt	
kg→jīn	… ▲ kg → jīn	
kg→lb.	… kg → lb.	
kg→liǎ	… ▲ kg → liǎng	
kg→oz.	… ▲ kg → oz.	
kg→scw	… kg → short cwt	
kg→sto	… kg → stone	
kg→sh.t	… ▲ kg → short ton	
kg→ton	… ▲ kg → ton	
kg→trz	… ▲ kg → troy oz.	
K_J		Constant (see pp. 113ff).
km→mi. km→nmi	U→ x: ▼ km → mile etc.	[6] (1) {2}; {1} → {2} Convert distances. See pp. 150ff.
KTYP?	INFO KTYP? s	[3+] (−1) { } → {1} Assumes a key code in the *RoV* specified (cf. KEY?), checks it, and returns its key type: • 0 … 9 if it corresponds to a digit 0 … 9, • 10 if it corresponds to ., E, or +/−, • 11 if it corresponds to or , • 13 if it corresponds to a *softkey*, and • 12 if it corresponds to any other key. Helps in user interaction with routines (see *OMS 3*).
K(m)	X.FN Ellipt K(m)	[4] (1) {2, 3} Returns the *complete elliptic integral of the 1st kind* for ***m*** = ***x***. See p. 317.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
L	Reserved *variable* for *register* L.	
(LASTx)	RCL L	[2] (−1) Recalls the last x used (see *OMS 1*). Note this command will be recorded as RCL L in routines.
lb	EXP lb x	[2] (1) {1, 2, 3, 8*, 9*, 10} ({1} → {2, 3}) Returns the logarithm of x for base 2.
lbft→Nm	U→ ▲ Nm ← lbf·ft	[4] (1) {2}; {1} → {2} Convert torques and forces. See pp. 150ff.
lbf→N	U→ F&p: N ← lbf	
LBL	LBL *label*	[2+] (0) Identifies programs and routines for executing and branching. Read about such *labels* and specifying them in *OMS 3*.
LBL?	TEST LBL? ***label***	[4+] (0) Tests for existence of the *label* specified, anywhere in program memory. See LBL.
lb.→kg	U→ m: kg ← lb.	[4] (1) {2}; {1} → {2} Converts masses. See pp. 150ff.
LCM	INTS LCM	[4] (2) {1; 10} Returns the *Least Common Multiple* of x and y.[27] This will always be positive.
LEAD.0	SF L	*System flag* (see pp. 127ff)
LEAP?	TEST LEAP?	[4] (0) {2, 6} Takes x as a *date* in the format selected (or a *real number* in corresponding format), extracts the year, and tests for a leap year.
lg	lg	[2] (1) {1, 2, 3, 8*, 9*, 10} ({1} → {2, 3}) Returns the logarithm of x for base 10.

[27] $\mathrm{LCM}(x;y) = \left| {}^{xy}/_{\mathrm{GCD}(x;y)} \right|$. See also *GCD*.

TN: *LCM* correspond à *PPCM* en français. / *LCM* entspricht *kgV* auf Deutsch.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
LgNrm$_p$ LgNrm$_{\triangle}$ LgNrm$_{\triangle}$ LgNrm^{-1}	PROB LgNrm: LgNrm$_p$ etc.	[5] (1) {2}; {1} → {2} *Log-normal distribution* with $\mu = ln\,\bar{x}_g$ specified in **I** and $\sigma = ln\,\varepsilon$ in **J**. See $\bar{x}_g$ and ε and pp. 269ff. LgNrm^{-1} returns ***x*** for a given probability ***p*** in **X**, with ***μ*** in **I** and ***σ*** in **J**.
LgNrm:	PROB LgNrm:	*Submenu*. See p. 144.
liă→kg	U→ m: ▲ kg ← liăng	[6] (1) {2}; {1} → {2} Converts masses. See pp. 150ff.
LinF	FIT ▼ LinF	[4] (0) Selects the linear fit model for L.R. Relevant also for CORR, x̂, and ŷ. See pp. 278ff.
lĭ→m	U→ x: ▲ m ← lĭ	[5] (1) {2}; {1} → {2} Converts distances. See pp. 150ff.
LJ	BIT ▲ LJ	[5] (1) {10} Left justifies the bit pattern ***x*** within its *word* size:[28] The *stack* will lift, placing the left-justified *word* in **Y** and the count of bit-shifts necessary to justify the *word* in **X**. **Example** for *word* size 8: 1 0110$_2$ LJ returns $x = 3$ and $y = 1011\ 0000_2$
L$_m$ L$_{mα}$	X.FN Orthog L$_m$ etc.	[4] (2) {2}; {1} → {2} *Laguerre polynomials* and *Laguerre's generalized polynomials*. See pp. 286f.
ln	ln	[1] (1) {2, 3, 8*, 9*, 10}; {1} → {2, 3} Returns the natural logarithm of ***x***.
lnβ	X.FN lnβ	[4] (2) {2, 3}; {1} → {2} Returns the natural logarithm of *Euler's* Beta function (see p. 100).

[28] LJ works as on *HP-16C*.

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
lnΓ	X.FN lnΓ PROB ▲ lnΓ	[4] (1) {2, 3}; {1} → {2} Returns the natural logarithm of $\Gamma(x)$ (see p. 101). Allows also for calculating really great factorials (see an example in *OMS 2*).
ln(1+x)	EXP ln 1+x	[3] (1) {2, 8*} For $x \approx 0$, this function returns a more accurate result for the fractional part than $\ln(x)$ does.
LoadAll	I/O LoadAll	[3] Restores either the entire backup from the *FD* `SAVFILES/wp43.sav` whereto it was written by SAVE, or the entire *state file* selected (see SAVEST; pick the file from a *DMCP* menu on your calculator or a corresponding menu on your *PC*). LOADALL calls LOADPR, LOADEQ, LOADR, LOADV, LOADΣ, LOADSS, and signals either **Backup restored** or **State file restored**.[29]
LoadEq	I/O LoadEq	[3] (0) Loads all equations,[30] overwriting identical equations present in *RAM*, and signals **Saved equations restored**.[29]
LoadPr	I/O LoadPr	[3] (0) Loads all programs.[30] Stores them behind the programs already in *RAM* and signals **Saved programs appended**.[29]
LoadR	I/O LoadR	[4] Recalls the *registers*[30] (incl. the *stack* and *local registers*), and signals **Saved global and local registers (with local flags) restored**.[29] All *registers* saved in the file will be recalled.
LoadSS	I/O LoadSS	[4] (0) Recovers the system settings,[30] i.e. the entire calculator *configuration* (user assignments, *system variables* and *flags* covered by RESET, see pp. 75 and 190) plus all global *user flags*, MyMENU, MyPFN, Myα, and all *user menus*. Signals **Saved system settings restored**.[29]

[29] See SAVE and SAVEST (p. 64) and *App. B*. If space in *RAM* does not suffice for restoring an error will be thrown.

[30] Source is either the backup file or the *state file* selected. Cf. LoadAll.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
LoadV	I/O LoadV	[4] (0) Recalls all user-defined *variables*[30] and signals **Saved user variables restored**.
LoadΣ	I/O LoadΣ	[4] (0) Recovers the statistics matrices STATS and HISTO as well as the *summation registers*[30] and signals **Saved statistic data restored**. Throws an error if no statistic data are allocated.[29]
LocR	P.FN LocR ***n***	[4+] (0) Allocates ***n*** *local registers* (≤ 99) and 32 *local flags* for the *current routine*. All start cleared. Any subsequent LOCR in the same routine will set the amount of *local registers* to the new number. Cf. *OMS 3*.
LocR?	INFO LocR?	[4] (−1) { } → {1} Returns the number of *local registers* currently allocated for the *current routine*.
LogF	FIT ▼ LogF	[4] (0) Selects the logarithmic fit model for L.R. Relevant also for CORR, x̂, and ŷ. See pp. 278ff.
Logis_{p} $\text{Logis}_{\blacktriangle}$ $\text{Logis}_{\blacktriangle}$ Logis^{-1}	PROB Logis: Logis_{p} etc.	[5] (1) {2}; {1} → {2} *Logistic distribution* with ***μ*** given in **I** and ***s*** in **J**. See pp. 269ff.
Logis:	PROB Logis:	*Submenu*. See p. 144.
$\log_x y$	EXP $\log_x y$	[2] (2) {1, 2, 3, 8*, 9*, 10}; ({1} → {2, 3}) Returns the logarithm of ***y*** for the base ***x***.
LOOP	LOOP	*Menu*. See p. 143.
LOWBAT		*System flag* (see pp. 127ff)
l_{PL}		Constant (see pp. 113ff).
L.INTS	CATALOG VARS L.INTS	*Submenu* of all *long integer variables* defined at execution time. See pp. 133ff.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
L.R.	FIT L.R.	[3] (−2) or (-3) { } → {2} Pushes the parameters a_2 (in Z), a_1 (in Y), and a_0 (in X) of the fit curve through the data points accumulated in STATS on the *stack*, according to the fit model chosen (see LINF, ORTHOF, EXPF, POWERF, LOGF, HYPF, ROOTF, PARABF, CAUCHF, and GAUSSF). If BESTF is set, L.R. picks the fit model returning maximum r^2 out of the models allowed (see p. 25). For a straight line, a_0 is its y-intercept and a_1 its slope. For forecasting, see $\hat{x}$ and $\hat{y}$ (note $\hat{x}$ may return 2 results). See pp. 278ff for more.
l.y.→m	U→ x: m ← l.y.	[4,7] (1) {2}; {1} → {2} Convert distances and volumes. See pp. 150ff.
ℓ→gal$_{UK}$	U→ V: ▲ ℓ → gal$_{UK}$	
ℓ→gal$_{US}$	U→ V: ▼ ℓ → gal$_{US}$	
ℓ→qt.	U→ V: ▼ ℓ → quart	
m^2→ac m^2→ac$_{US}$	U→ A: m^2 → acre etc.	[5,6] (1) {2}; {1} → {2} Convert areas and volumes. See pp. 150ff.
m^2→ha	U→ A: m^2 → ha	
m^2→mŭ	U→ A: m^2 → mŭ	
m^3→bbl	U→ V: m^3 → barrel	
m^3→fz$_{UK}$ m^3→fz$_{US}$	U→ V: m^3 → floz$_{UK}$ etc.	
m^3→gl$_{UK}$ m^3→gl$_{US}$	U→ V: m^3 → gal$_{UK}$ etc.	

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
MAD	STAT ▲ MAD	[4] (−2) { } → {2} Pushes the *median absolute deviations* of the y- and x-data accumulated on the *stack*.[31] $\mathrm{M_{AD}}(x) := \mathrm{median}\lvert x_i - \mathrm{median}(x_i)\rvert$
MANT	PART MANT	[3] (1) {2}; {1} → {2} Returns the mantissa m of $x = m \cdot 10^h$. Cf. EXPT.
MASKL	BIT MASKL *n*	[3+] (−1) { } → {10} Create a bit mask.[32] The mask length (or its address) follows the command. Thus, the mask is pushed on the *stack*. MASKL **0** and MASKR **0** return 0. **Example:** For WSIZE 8, MASKL 3 will return a mask *word* 1110 0000₂. Use it, e.g., for extracting the top 3 bits of an arbitrary *byte* via AND.
MASKR	BIT MASKR *n*	
MATRS	CATALOG VARS MATRS	*Submenu* of all matrix *variables* defined at execution time. See pp. 133f.
MATR?	TEST ▲ MATR?	[4] (0) Checks if x is a *real* or *complex* matrix.
MATX	MATX	*Menu*. See p. 143.
Mat.A	Reserved matrix *variables*	for solving systems of linear equations (see *OMS 2*).
Mat.B		
Mat.X		
Mat_A	MATX SIM EQ Mat A etc.	[4] (−1) Creates, if necessary, and opens the coefficient matrix **Mat.A** (or the constant vector **Mat.B**) in a system of linear equations for editing (see *OMS 2*).
Mat_B		
Mat_X		[4] (−1) Returns the solution vector **Mat.X** and exits M.SIMQ

[31] Compare $s^2 = E([x_i - E(x_i)]^2)$.

[32] Work like MASKL and MASKR on *HP-16C*, but there the mask length is taken from **X**.

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
max	X.FN max	[4] (2) {1, 2, 4, 5, 6, 7, 10} Returns the maximum of x and y.
MDY		*System flag* (see pp. 127ff)
m_e		Constant (see pp. 113ff).
MEDIAN	STAT ▲ MEDIAN	[4] (−2) { } → {2} Returns the *medians* of the y- and x-data accumulated, pushing those on the *stack*. The *median* cuts the data set in equal halves.
MEM?	INFO MEM?	[3] (−1) { } → {1} Returns the number of free *bytes* in *RAM* (i.e. the pool size), taking into account all *registers* currently allocated and their current contents.
MENU	P.FN P.FN2 MENU	Displays the programmable *menu*. See *OMS 3*.
MENUS	CATALOG MENUS	*Submenu* of all *menus* defined at execution time. See pp. 133ff.
MIL	MODE MIL	[3] (0) Sets the *ADM* to *mil*. See *OMS 2*.
MIL→	∡→ MIL→	[4] (1) {1, 2, 4} → {4} Convert *angles* as described on pp. 131f.
MIL→D	∡→ ▲ MIL→D	
MIL→R	∡→ ▲ MIL→R	
min	X.FN min	[4] (2) {1, 2, 4, 5, 6, 7, 10} Returns the minimum of x and y.
MIRROR	BIT MIRROR	[4] (1) {10} Reflects the bit pattern in x (e.g., for *word* size 8, 0001 0111_2 MIRROR will return 1110 1000_2).
mi.→km	U→ x: ▼ km ← mile	[6,5] (1) {2}; {1} → {2} Convert distances. See pp. 150ff.
mi.→m	U→ x: m ← mile	
mℓ→cp_I	U→ V: ▲ ▲ mℓ → cup_I	[7,6] (1) {2}; {1} → {2} Convert kitchen volumes. See pp. 150ff.
mℓ→cp_{UK}	... ▲ mℓ → cup_{UK}	
mℓ→cp_{US}	... ▼ mℓ → cup_{US}	

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
mℓ→fz$_{UK}$	... ▲ mℓ →floz$_{UK}$	
mℓ→fz$_{US}$	... ▼ mℓ →floz$_{US}$	
mℓ→pt$_{UK}$	... ▲ mℓ →pint$_{UK}$	
mℓ→pt$_{US}$	... ▼ mℓ →pint$_{US}$	
mℓ→tb$_{I}$	... ▲ ▲ mℓ →table spoon$_{I}$	
mℓ→tb$_{UK}$	... ▲ mℓ →tbsp$_{UK}$	
mℓ→Tb$_{US}$	... ▼ mℓ →Tbsp$_{US}$	
mℓ→ts$_{I}$	... ▲ ▲ mℓ → tea spoon$_{I}$	
mℓ→ts$_{UK}$	... ▲ mℓ → tsp$_{UK}$	
mℓ→ts$_{US}$	... ▼ mℓ → tsp$_{US}$	
mmHg→Pa	U→ F&p: Pa ← mmHg	[5] (1) {2}; {1} → {2} Convert pressures, and heights. See pp. 150ff.
mm→in.	U→ x: ▼ mm → inch etc.	
mm→pt.		
m$_n$		Constants (see pp. 113ff).
m$_n$/m$_p$		
MOD	MOD	[2] (2) {1, 2, 10} Returns $y \bmod x$ (see *OMS 2* for examples). Cf. RMD.
	INTS MOD	
MODE	MODE	*Menu*. See p. 143.
MONTH	CLK MONTH	[4] (1) {2, 6} → {1} Takes x as a *date* in the format selected (or a *real number* in corresponding format) and extracts the month.
m$_p$		Constants (see pp. 113ff).
m$_p$/m$_e$		
m$_{PL}$		

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
MSG	P.FN P.FN2 MSG ***n***	[5+] (0) {1, 2} Throws just the (*temporary*) *error message* specified by the error code ***n*** (see *App. C* on pp. 206ff). Compare ERR.
m_u m_uc^2		Constants (see pp. 113ff).
MULT×		*System flag* (see pp. 127ff)
MULπ	MODE MULπ	[3] (0) Sets the *ADM* to *multiples of* π.
MULπ→	↙→ MULπ→	[4] (1) {1, 2, 4} → {4} Converts *angles* as described on pp. 161f.
mŭ→m^2	U→ A: m^2 ← mŭ	[5] (1) {2}; {1} → {2} Converts areas. See pp. 150ff.
MVAR	P.FN MVAR ***name***	[3+] (0) Defines a *menu variable* (such *variables* are required for VARMNU). Works in *PEM* and running programs only. See *OMS 3*.
MyMENU	CAT. MENUS MyMENU	User *menu*.
MyPFN	CAT. MENUS MyPFN	User *menu* in *PEM*.
Myα	CAT. MENUS Myα	User *menu* in *AIM*.
m_μ		Constant (see pp. 113ff).
Mπ→R	↙→ Mπ→R	[4] (1) {1, 2, 4} → {4} Converts *angles* as described on pp. 161f.
M.CMAX M.CMIN	MATX ▲ CMAX etc.	[5] (-2) {8} → {1 or 2 in **X**, 1 in **Y**} Finds the greatest (or smallest) element in column ***j*** of the *indexed matrix* (cf. INDEX); pushes its row number ***i*** and its value a_{ij} on the *stack*. If two elements a_{ij} are equal, the one with the greater ***i*** will be returned. If you want to search more than one column (e.g., to find the smallest element of an entire matrix) you may re-dimension the matrix, search it, then re-dimension it back again (see M.DIM).

See *OMS 6.*

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
M.DELR	DELR with M.EDIT displayed	[2] (0) {8, 9} Deletes the row of matrix elements where the cursor is. Will not work for a matrix with only 1 row.
M.DIM	MATX DIM ***name***	[4+] (0) {1, 2} Creates a new matrix[33] or re-dimensions an existing matrix to IP(***y***) rows and IP(***x***) columns (size limited by memory available).
M.DIM?	INFO DIM? MATX ▲ DIM?	[4] (-2) {8, 9} → {1} Returns the dimensions of the matrix ***x*** (rows to **Y**, columns to **X**). Note the matrix is saved in **L**. Former ***y*** goes into **Z**, former ***z*** into **T**, etc.
M.DY	CLK ▲ M.DY	[4] (0) Selects the format `mm/dd/yyyy` for *dates*.
M.EDI	MATX EDIT	[3] {8, 9} Opens ***x*** using the *Matrix Editor*. [34]
M.EDIN	MATX EDITN ***name***	[4+] (2) Opens a matrix[33] using the *Matrix Editor*.[34]
M.EDIT		*Submenu* for matrix editing, called by M.EDI and M.EDIN. See p. 143.
M.FIND	MATX ▲ FIND	[5] (-1) {8, 9} Searches the *indexed matrix* for the value ***x***. If it is found, ***i*** and ***j*** will point to the element (else they stay unchanged) and M.FIND will … • return `Yes` (else `No`) in manual *RUM*; • execute the next step in a program running, else skip it.

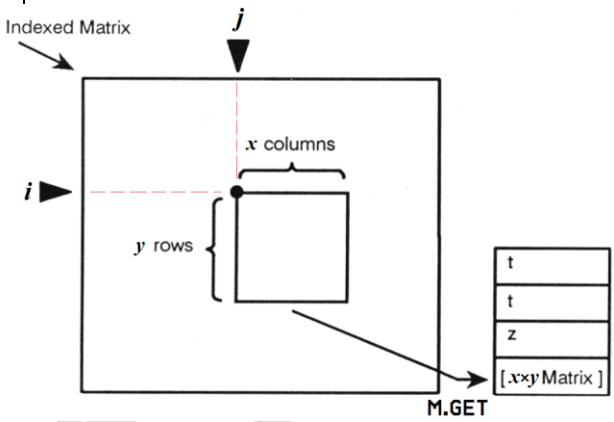
[33] Your WP43 allows for matrices in *variables*, but also in *registers*. A named matrix was explicitly required on the *HP-42S* only. Cf. DIM in the *HP-42S OM*, p. 217.

[34] See *OMS 2*. M.EDI (labeled EDIT in MATX) and M.EDIN (labeled EDITN) disable *ASL*.

In *HP-42S*, both do not actually disable *ASL*; they preserve the *stack lift* state – you can observe this if you execute ENTER vs. a *stack-lift*-enabling operation (e.g., x⇄y) just before invoking them. This behavior is not really useful; it was dropped here since *HP-42S* compatibility is not required.

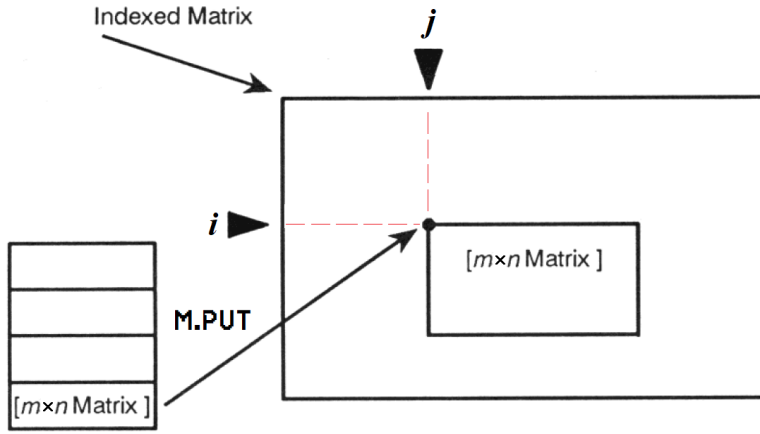
DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
M.GET	MATX **GETM**	[4] (0) {1, 2} → {8, 9} Gets a sub-matrix with IP(y) rows and IP(x) columns out of the *indexed matrix*, starting with its element ***i j***, into **X**. See the picture and M.PUT.



M.GOTO	**GOTO**	with M.EDIT displayed	[2] (0) Asks for target row and column and goes to this matrix element, updating the index pointers ***i*** and ***j***.
M.GROW	**GROW**		[2] (0) Sets GROW to allow the *indexed matrix* to grow automatically (cf. J+ and *OMS 2*). See M.WRAP.
M.INSR	**INSR**		[2] (0) Inserts a new row of elements containing 0., above the current cursor position in the matrix.
M.LU	MATX ▲ **M.LU**		[5] (-2) {8, 9} Computes the *LU* (a.k.a. *LR*) *decomposition* of the matrix ***x*** and returns the pivot matrix in **Z**, the lower (left) triangular matrix ***L*** in **Y**, and the upper (right) triangular matrix ***U*** in **X**. *L* is computed such that all its diagonal elements are 1. Principle: [35] $x = \begin{bmatrix}1 & 2 & 3\\1 & 1 & 1\\3 & 3 & 1\end{bmatrix} = \begin{bmatrix}1 & 0 & 0\\1 & 1 & 0\\3 & 3 & 1\end{bmatrix} \cdot \begin{bmatrix}1 & 2 & 3\\0 & -1 & -2\\0 & 0 & -2\end{bmatrix} = L \cdot U$

[35] See https://en.wikipedia.org/wiki/LU_decomposition (or equivalent articles in other languages) for more, also about pivoting.

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
		You can press [x] [x] after M.LU to get the original matrix back or recall it from **L**.
M.NEW	MATX NEW	[3] (2) {1, 2} → {8} Creates a new matrix (size limited by memory available). Its number of rows is supplied as IP(***y***) and its number of columns as IP(***x***). M.NEW will return a matrix ***x*** with all elements set to zero.
M.PUT	MATX PUTM	[4] (0) {8, 9} Puts the matrix ***x*** as it is into the *indexed matrix*, starting with the target element a_{ij}. The size of the target matrix will not change. Cf. M.GET.
		
M.QR	MATX ▲ M.QR	[5] (-1) {8, 9} Computes the *QR decomposition* of the matrix ***x*** and returns the unitary matrix ***Q*** in **Y** and the upper right triangular matrix ***R*** in **X**.
M.R⇄R	MATX ▲ R⇄R	[5] (0) {1, 2} Swaps rows IP(***x***) and IP(***y***) of the *indexed matrix*.
M.SIMQ		*Submenu* of MATX, called by SIM_EQ.
M.SQR?	TEST ▲ M.SQR?	[5] (0) Returns true if ***x*** is a square matrix.

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
M.WRAP	WRAP	[2] (0) Controls the matrix index pointers (see *OMS 2*). Clears GROW. Cf. M.GROW.
M.→	→	[1] (0) Browse the matrix ***x***. Disable *ASL*.
M.←	←	
M.↑	↑	
M.↓	↓	
m:	U→ m:	*Submenu* of mass conversions. See p. 145.
m→au	U→ x: m → au	[4,6,5] (1) {2}; {1} → {2} Convert distances. See pp. 150ff.
m→chǐ	… ▲ m → chǐ	
m→cùn	etc.	
m→fēn		
m→ftm.	… ▲ m → fathm	
m→ft.	… m → foot	
m→in.	… m → inch	
m→ft$_{US}$	… ▼ m → surv. foot$_{US}$	
m→lǐ	… ▲ m → lǐ	
m→l.y.	… m → l.y.	
m→mi.	… m → mile etc.	
m→nmi		
m→pc	… m → pc	
m→pt.	… m → point	
m→yd.	… m → yard	
m→yǐn	… ▲ m → yǐn	
m→zhàng	… ▲ m → zhàng	
$M_{\odot}$		Constants (see pp. 113ff).
$M_{\oplus}$		
N_A		Constants (see pp. 113ff).

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
NaN		
NAND	BIT NAND	[4] (2) Works in analogy to AND. Cf. p. 22.
NaN?	TEST ▲ NaN?	[5] (0) Returns `true` if x is *Not a Number*. For *complex* x, Re(x) or Im(x) must be ±NaN..
nBINS	FIT HISTOG nBINS STAT HISTOG nBINS	[4] (1) Sets the number of bins for a histogram to be drawn.
$NBin_p$ $NBin_▲$ $NBin_▲$ $NBin^{-1}$	PROB NBin: $NBin_p$ etc.	[5] (1) {2}; {1} → {2} Negative binomial distribution with the number of successes ***k*** in **X**, the gross probability p_0 of a success in a single draw in **I**, and the number of failures ***n*** until the experiment is stopped in **J**. See pp. 269ff.
NBin:	PROB NBin:	*Submenu*. See p. 144.
NEIGHB	INFO NEIGHB	[4] (2) {1} Returns • $x + 1$ for $x < y$, • x for $x = y$, and • $x - 1$ for $x > y$.
		(2) {2} Returns the nearest machine-representable number to x in the direction towards y in the mode set. For … • $x < y$, it will be the machine successor of x; • $x = y$, it will be y; • $x > y$, it will be the machine predecessor of x. It may be useful investigating numeric stability.[36]

[36] NEIGHB here corresponds to NEIGHBOR in the *HP-71 Math Pac*.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
NEXTP	X.FN NEXTP	[4] (1) {1, 10}; {2} → {1} Returns the next prime number greater than |IP(x)|. See also PRIME? on p. 71. Searching for the next prime number may take long and thus can be aborted by EXIT. **ATTENTION:** See p. 195 for limitations.
nmi.→km	U→ x: ▼ km ← nmi.	[6,5,4] (1) {2}; {1} → {2} Convert distances and torques. See pp. 150ff.
nmi.→m	U→ x: m ← nmi.	
Nm→lbf·ft	U→ ▲ Nm → lbf·ft	
NOP	P.FN P.FN2 NOP	'Empty' program step (for historical reasons only).
NOR	BIT NOR	[4] (2) Works in analogy to AND. Cf. p. 22.
Norml_p $\text{Norml}_{\triangle}$ $\text{Norml}_{\triangle}$ Norml^{-1}	PROB Norml: Norml_p etc.	[4] (1) {2}; {1} → {2} *Normal distribution* with an arbitrary mean $\boldsymbol{\mu}$ given in $\mathbf{I}$ and an *SD* $\boldsymbol{\sigma}$ in $\mathbf{J}$. See pp. 269ff and *OMS 2* for an application example. Norml^{-1} returns $\boldsymbol{x}$ for a given probability $\boldsymbol{p}$ in $\mathbf{X}$, with $\boldsymbol{\mu}$ in $\mathbf{I}$ and $\boldsymbol{\sigma}$ in $\mathbf{J}$.
Norml:	PROB Norml:	*Submenu*. See p. 144.
NOT	BIT NOT	[3] (1) {10} Inverts x bit-wise (as on *HP-16C*).
		(1) {1, 2} → {1} Returns 1 for $x = 0$, and 0 for $x \neq 0$.
NPER	FIN TVM n_{PER}	Reserved *integer variable* for the total number of • payment periods for your loan or • compounding periods for your investment.
NUM.IN		*System flag* (see pp. 127ff)
NXTFIT	FIT ASSESS NXTFIT	[5] (0) Plots the next fit model allowed together with the accumulated data points. Cf. BESTF on p. 25.

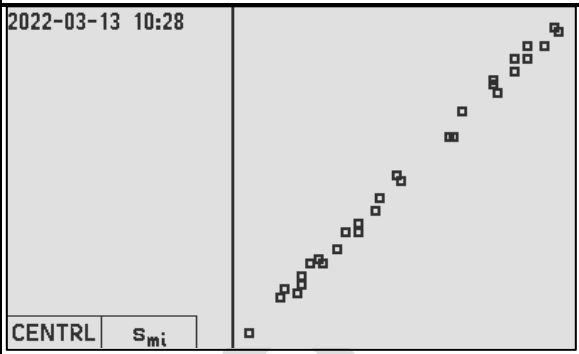
DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
nΣ	Σ n	[3] (−1) { } → {1} Recalls the number of accumulated data points.
N→lbf	U→ F&p: N → lbf	[4] (1) {2}; {1} → {2} Converts forces. See pp. 150ff.
ODD?	TEST ODD?	[4] (0) Checks if x is integer and odd (cf. INT? on p. 48). Note *reals* with FP(x) = 0 count as integers as well.
OFF	OFF	[2] (0) Turns your *WP43* off. In *PEM*, inserts a step to turn it off under program control. A timer running in background will continue!
OR	BIT OR	[3] (2) Works in analogy to AND, cf. p. 22.
OrthoF	FIT ▼ OrthoF	[4] (0) Selects the linear fit model for orthogonal regression. Relevant for CENTRL, CORR, L.R., s_{mi}, $\hat{x}$, and $\hat{y}$. See pp. 278ff.
ORTHOG	X.FN Orthog	*Submenu* for orthogonal polynomials, see p. 146.
OVERFL		*System flag* (see pp. 127ff), set by SF B (big).
oz.→g oz.→kg	U→ m: kg ← oz. etc.	[4] (1) {2}; {1} → {2} Convert masses. See pp. 150ff.
p_0		Constant (see pp. 113ff).
ParabF	FIT ▼ ParabF	[5] (0) Selects the parabolic fit model for L.R. Relevant also for CORR, $\hat{x}$, and $\hat{y}$. See pp. 278ff.
PART	PART	*Menu*. See p. 143.
PAUSE	P.FN PAUSE ***n***	[3+] (0) Within a routine running, refreshes the *LCD* and pauses program execution for n ticks (see TICKS on p. 90), with $0 \le n \le 99$. This pause will terminate early when you press a key. Cf. KEY?
Pa→atm Pa→bar Pa→iHg	U→ F&p: Pa → atm ... Pa → bar ... Pa → in.Hg	[5,4] (1) {2}; {1} → {2} Convert pressures and distances. See pp. 150ff.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
Pa→mmHg	... Pa → mmHg	
Pa→psi	... Pa → psi	
Pa→torr	... Pa → torr	
pc→m	U→ x: m ← pc	
PERM	PROB P_{yx}	[3] (2) {1} Returns the number of possible arrangements (a.k.a. *permutations*) of ***x*** *objects* taken out of a set of ***y*** *objects*. No *object* shall occur more than once in an arrangement, and different orders of the same ***x*** *objects* are counted separately. Cf. COMB.
		(2) {2, 3} See pp. 267f for the formula.
PER/a	FIN TVM per/a	Reserved *integer variable* for the annual number of • payments for your loan or • compounding periods of your investment.
PGMINT	ADV PGMINT ***label***	[4+] (0) Specifies the address of the routine containing the expression to be integrated or solved, respectively. See *OMS 4*.
PGMSLV	ADV PGMSLV ***label***	
PIXEL	P.FN P.FN2 PIXEL	[5] (0) Turns on a single pixel (dot) on the *LCD*. The location of the pixel is given by IP(***x***) and IP(***y***). Cf. AGRAPH on p. 21.
PLAY	I/O PLAY ***s***	[4+] (0) Plays a ***n***×2 or ***n***×3 matrix ***s*** of buzzer tones: each row contains one tone with its frequency in Hz, duration in ms, and (optionally) the volume level (1…11).
PLOT	FIT PLOT	[4] (0) Opens a new screen and plots the data points stored in STATS for *MSA* (see pp. 294ff). Cf. CENTRL on p. 28. Find examples in *OMS 2*.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
		
PMT	FIN TVM PMT	Reserved *real variable* for the payment per period of your investment or loan in *TVM*. See *OMS 5.*
P_n	X.FN Orthog P_n	[4] (1) {2}; {1} → {2} *Legendre polynomials*. See pp. 286f.
POINT	P.FN P.FN2 POINT	[5] (1) {1, 2} Turns on a square point (3×3 *pixels*) on the *LCD*. The position of its center is given by IP(***x***) and IP(***y***). Cf. AGRAPH on p. 21.
$Poiss_p$ $Poiss_{\blacktriangle}$ $Poiss_{\blacktriangle}$ $Poiss^{-1}$	PROB Poiss: $Poiss_p$ etc.	[5] (1) {2}; {1} → {2} *Poisson distribution* with the number of successes ***g*** in **X** and the *Poisson* parameter ***λ*** in **I**. See pp. 269ff. $Poiss^{-1}$ returns the maximum number of successes ***m*** for given probability ***p*** in **X** and ***λ*** in **I**.
Poiss:	PROB Poiss:	*Submenu*. See p. 144.
POLAR		*System flag* (see pp. 127ff), set by SF X.
PopLR	P.FN PopLR	[4] (0) Pops the *local registers* allocated to the *current routine* (see *OMS 3*) without returning to the caller. See LOCR (p. 56) and RTN (p. 79).
PowerF	FIT ▼ PowerF	[4] (0) Selects the power curve fit model for L.R. Relevant also for CORR, x̂, and ŷ (see pp. 278ff).

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
PRIME?	TEST PRIME?	[4] (0) {1, 2, 10} Checks if |IP(x)| is a prime number. Returns `true` for prime and `false` for composite. **ATTENTION:** See p. 195 for limitations.
PROB	PROB	*Menu*. See p. 144.
PROG		*Submenu* of *global labels* defined when calling XEQ etc. See *OMS 3*.
PROGS	CATALOG PROGS	*Submenu* of *global labels* defined at execution time. See pp. 133f.
PROMPT	P.FN PROMPT **s**	[4+] (0) Displays ***s*** left-adjusted in the top numeric row (and below if it needs more space) and stops program execution, lighting ❶. Cf. AVIEW.
PROPFR		*System flag* (see pp. 127ff)
pr→dB	U→ ▲ dB ← power ratio	[4,5,7] (1) {2}; {1} → {2} Convert ratios, pressures, print heights, and kitchen volumes. See pp. 150ff.
psi→Pa	U→ F&p: Pa ← psi	
pt.→m	U→ x: m ← point etc.	
pt.→mm		
pt_{UK}→mℓ	U→ V: ▲ mℓ ← $pint_{UK}$	
pt_{US}→mℓ	U→ V: ▼ mℓ ← $pint_{US}$	
PUTK	P.FN PUTK **s**	[4+] (0) Assumes a key code in ***s***. Stops program execution, takes said code and puts it in the keyboard buffer resulting in immediate execution of the corresponding call. R/S is required to resume program execution then. May help in user interaction with routines (see *OMS 3*).[37]
PV	FIN TVM PV	Reserved *real variable* for the present value of your investment or loan in *TVM*. See *OMS 5*.

[37] PUTK was invented for the *WP 34S*.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
P.FN	P.FN	*Menu* of functions for programming. See p. 144.
P.FN2	P.FN P.FN2	*Submenu*. See p. 144.
P:	U→ P:	*Submenu* of power conversions. See p. 150.
Q_3-Q_1	STAT ▲ Q_3-Q_1	[4] (−2) { } → {2} Pushes the distance between the *75%*- and *25%*-*percentile* (a.k.a. 3rd and 1st *quartile*) of the y- and x-data accumulated on the *stack*. Note this may only make sense for data sets containing at least 12 samples.
QUIET		*System flag* (see pp. 127ff)
qt.→ℓ	U→ V: ▲ ▲ ℓ ← quart	[8] (1) {2}; {1} → {2} Converts volumes. See pp. 150ff.
R		Constant (see pp. 113ff).
(r)		Cf. CORR on p. 32.
RAD	MODE RAD	[3] (0) Sets the *ADM* to *radians*.
RAD→	∡→ RAD→	[4] (1) {1, 2, 4} → {4} Converts *angles* as described on pp. 161f.
RANGE	DISP ▲ RANGE	[5] (0) Limits the range of *real numbers* or parts to be displayed or entered to $\pm 10^{\pm n}$ with n = IP(x), $99 \leq n \leq 6145$. For greater x, n will be set to 6145; for less it will be set to 99. *Startup default* is **999**. In consequence, '0.' will be displayed for smaller values and '∞' for greater, saving screen space for sensible data. Cf. HIDE (p. 46); see also pp. 195ff.
RANGE?	DISP ▲ RANGE? INFO ▲ RANGE?	[4] (-1) { } → {1} Returns the current range setting.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
RANI#	PROB ▲ RANI#	[4] (−1) { } → {1} Returns a pseudo random *integer* n with $\mathrm{IP}[\min(x,y)] \le n \le \mathrm{IP}[\max(x,y)]$.[38] After executing RANI#, the *stack* will look like [n, x, y, …], so you can roll down n and call RANI# again to get another random *integer* within the same limits. Use RANI# for throwing dice, for instance.
RAN#	PROB ▲ RAN#	[4] (−1) { } → {2} Returns a pseudo random *real number* $x \in [0\,;1)$.[38] See also SEED (on p. 82) and p. 332.
RBR	RBR	[2] Calls the *register browser* – see *OMS 5*. RBR works also in *PEM* but it is not programmable. In RBR, numbers are generally displayed in their format chosen. Since it uses the small font always, more digits can be shown than in a standard numeric row. For *reals* and ALL **0**, RBR display will turn to SCI or ENG at 33 digits in worst case. Extended display precision may be observed for *complex numbers* as well.
RCL	RCL s	[1+] (−1) Recalls (copies) the content of the *RoV* specified and pushes it on the *stack*.
RCLCFG	RCL Config s	[3+] (0) Recalls a calculator *configuration* stored by STOCFG (see *OMS 2* and *6*) from the *RoV* specified. See also RESET.
RCLEL	MATX RCLEL RCL ...EL OLD with M.EDIT displayed	[4,3,1] (−1) { } → {2, 3} Recalls a copy of the current element a_{ij} of the *indexed matrix*. Cf. STOEL.

[38] For equal seeds, you will get equal sequences of 'random' numbers. But see SEED.

ATTENTION: Undoing RAN# or RANI# will return previous stack but not reset current seed.

Note you can easily produce pseudo random numbers following a normal distribution with mean 5 and *SD* 1.826 by adding 10 returns of RAN# for each such number, for instance. This is brute force. More elegant may be using the *Box-Muller transform* (see https://en.wikipedia.org/wiki/Box%E2%80%93Muller_transform) or the *Marsaglia polar method* (https://en.wikipedia.org/wiki/Marsaglia_polar_method).

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
RCLIJ	MATX ▲ RCLIJ RCL ...IJ	[4,3] (−2) { } → {1} Recalls the current values of the matrix index pointers into **X** (= column number) and **Y** (= row number). If no matrix is indexed, an error is thrown. Cf. STOIJ.
RCLMAX	RCL max *s* RCL ▲ *s*	[2+] (1) {1, 2, 4, 5, 6, 10} Replaces x with the maximum of s and x.[39]
RCLMIN	RCL min *s* RCL ▼ *s*	[2+] (1) {1, 2, 4, 5, 6, 10} Replaces x with the minimum of s and x.[39]
RCLSTK	RCL Stack *s*	[3+] Recalls 4 or 8 values from a set of *registers* starting at address s, and pushes them on the *stack*. This is the inverse command of STOSTK.
RCL+	RCL + *s*	[2+] (1) Recalls the content of the *RoV* addressed, executes the operation specified, and puts the result on the *stack* like a *monadic* function.[39] **Example:** RCL− 12 replaces x with $x - r12$.
RCL−	RCL − *s*	
RCL×	RCL × *s*	
RCL/	RCL / *s*	
RDP	DISP RDP *n*	[4+] (1) {2, 3, 4, 5, 8*, 9*} Rounds x to n decimal places ($0 \le n \le 99$, think of FIX format), taking the RM setting into account. See RM and compare RSD. **Examples:** 123.456 789 012 RDP **5** returns 123.456 79 (5 decimals survive). $1.234\,56\times10^{-50}$ RDP **53** returns 1.235×10^{-50} (53 decimals survive).
Re	CPX Re PART Re	[3] (1) {2, 3} → {2}; {9} → {8} Returns the *real* part of x. Cf. IM.
r_e		Constant (see pp. 113ff).

[39] Only legal operations according to the *DT* matrices in *OMS 2* will work. See also the examples given there.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
READP	I/O READP ***name***	[4+] (0) Reads a named program file from the *FD* and puts it in program memory (if a program carrying an identical name is found there, it will be overwritten). See WRITEP (p. 94) and *OMS 3.*
REALDF		Reserved *integer variable* for *real number* display format: 0: ALL, 1: FIX, 2: SCI, 3: ENG.
REALS	CATALOG VARS REALS	*Submenu* of all *real variables* defined at execution time. See pp. 133f.
REAL?	TEST ▲ REAL?	[4] (0) Checks if x is of *DT 2* or *8* (cf. INT? on p. 48).
REDRAW	EQN Solver reDraw	[4] (0) Redraws the graph for the current equation as it was looking immediately after DRAW (cf. p. 36), undoing previous ZOOMs this way.
REM	P.FN REM	[4] In *PEM*, inserts a program step containing a comment string. Such a string may be up to 140 *characters* long and take up to 4 lines. REM is skipped when the program is running. Does nothing in *RUM*.
RESET	CLR ▲ RESET	[4] Executes CLALL and resets your *WP43* to *SDC*, i.e. to 2COMPL, ALL **0**, DEG, DENMAX **0**, DSTACK **4**, GAP **3**, J/G **1752.0914**, LinF, LocR **0**, RM **0**, RSD **34**, TDISP **0**, and WSIZE **64**. It also sets RANGE to **999**, and sets ASLIFT, DECIM., DENANY, ENDPMT, MULT×, PROPFR, SSIZE8, TDM24, and YMD; it clears all other *system flags* and resets the random number generator. See said commands, *system flags*, and p. 190.
(RESET)	in TIMER	See T.RES.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
RE→CX	CC (works in *RUM* only)	[1,4] (2) {2} → {3} Composes a *complex number* out of two *reals* or *integers* x and y, setting ℂ and taking either… • (for ⌞) the *real* part from **Y** and *imaginary* part from **X**, or • (for ⊙) magnitude from **Y** and phase from **X**.
	CPX RE→CX	(2) {8} → {9} Works in analogy for two *real* matrices x and y.
Re⇄Im	CPX Re⇄Im	[3] (1) {3, 9*} Swaps *real* and *imaginary* parts of a *complex* x.
RJ	BIT ▲ RJ	[5] (1) {10} Right justifies the bit pattern x within its word size, in analogy to LJ (cf. p. 54). The *stack* will lift, placing the right-justified *word* in **Y** and the count of bit-shifts necessary to justify the *word* in **X.** **Example:** $10\ 1100_2$ RJ results in $y = 1011_2$ and $x = 2$.
R_K		Constant (see pp. 113ff).
RL	BIT ▲ RL *n*	[5+] (1) {10} Rotates the bit pattern in **X** left n (≤ 63) bits.[40]
RLC	BIT ▲ RLC *n*	[5+] (1) {10} Rotates the bit pattern in **X** left n (≤ 63) bits through CARRY.[40]
RM	MODE RM *n*	[4+] (0) Sets floating point rounding mode. This mode is used only for RSD or when converting from the extended precision internal format (typically 39 digits) to packed *reals*. It will not alter the display nor change the behavior of ROUND. The following seven modes are supported:

[40] RL **0** and RLC **0** execute as NOP, but load **L**. See *OMS 2*.
RL and RLC work like RLn and RLCn on the *HP-16C*, respectively.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
		0: round half even: ½E 0.5 rounds to next even number (default, used in science). **1**: round half up: ½↑ 0.5 rounds up ('business-man's rounding' [41]). **2**: round half down: ½↓ 0.5 rounds down. **3**: round up: ←0→ rounds away from 0. **4**: round down: →0← rounds towards 0. **5**: ceiling: ⌈x⌉ rounds towards + ∞. **6**: floor: ⌊x⌋ rounds towards – ∞. The abbreviations printed on grey here are used in STATUS for indicating the respective rounding mode (see *OMS 5*).
RMD	INTS RMD	[4] (2) {1, 2, 10} Returns the remainder of the division ***y / x***.[42] See *OMS 2* for examples. Cf. MOD.
R_{Moon}		Constant (see pp. 113ff).
RM?	INFO RM?	[3] (−1) { } → {1} Returns the floating point rounding mode. Cf. RM.
RNORM	MATX ▲ RNORM	[5] (1) {8, 9} → {2} Calculates the row norm of the matrix ***x***, i.e. the maximum value (over all rows) of the sums of the absolute values of all elements in a row.[43] For a vector, the row norm is the largest absolute value of any of its elements. Compare ENORM.
RootF	FIT ▼ RootF	[5] (0) Selects the root fit model for L.R.; relevant also for CORR, x̂, and ŷ. See pp. 278ff.

[41] *TN*: Cela correspond à l'arrondi commercial. / Das entspricht kaufmännischer Rundung.

[42] Corresponds to RMD on *HP-16C* but works for *reals* on your *WP43* as well.

[43] This works like RNRM on *HP-42S*.

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
ROUND	DISP **ROUND**	[3] (1) {2, 3, 4, 5, 6, 8*, 9*} Rounds ***x*** using the current display format; i.e. all digits not currently displayed will be lost for further calculations, just the rounded value survives.[44]
ROUNDI	DISP **ROUNDI**	[3] (1) {1, 8*}; {2} → {1} Rounds ***x*** to next *integer*. ½ rounds to 1. Cf. IP.
RR	BIT ▲ **RR** ***n***	[5+] (1) {10} Rotates the bit pattern in **X** right ***n*** (≤ 63) bits.[45]
RRC	BIT ▲ **RRC** ***n***	[5+] (1) {10} Rotates the bit pattern in **X** right ***n*** (≤ 63) bits through CARRY.[45]
RSD	DISP **RSD** ***n***	[4+] (1) {2, 3, 4, 5, 8*, 9*} Rounds ***x*** to ***n*** significant digits (1 ≤ ***n*** ≤ 34), taking the RM setting into account. Think of SCI. Cf. RM and RDP. **Example:** 123.456 789 012 RSD **7** returns 123.456 8 (7 digits survive).
RSUM	MATX ▲ **RSUM**	[5] (1) {8, 9} Calculates the row sum of the matrix ***x***, returning an ***m*** × 1 matrix filled with the signed row sums of the ***m*** × ***n*** input matrix.

[44] This works like RND on *HP-42S*.

[45] RR **0** and RRC **0** execute as NOP, but load **L**. See *OMS 2*.
RR and RRC work like RRn or RRCn on *HP-16C*, respectively.

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
RTN	RTN	[2] (0) In *PEM*, RTN shall be the logically last command in a routine (see *OMS 3*). In a routine executing (e.g., **Y**), RTN pops *local* data (cf. POPLR) and returns to the caller, i.e. moves the *PP* to the step just behind XEQ '**Y**'. If there is none (i.e. the routine is top level), program execution will halt and the *PP* will be set to step 0000. If pressed in *RUM* with no routine executing, RTN resets the *PP* to the first step of *current program* (see *OMS 3*).
RTN.SKP	P.FN P.FN2 RTN.SKP	[4] (0) Works like RTN but in a routine executing, RTN.SKP moves the *PP* two steps behind the XEQ that called said routine.
RUNIO RUNTIM		*System flags* (see pp. 127ff)
R-CLR	P.FN R-CLR	[4] (0) {2} Interprets x in the form `sss.nn`. Clears ***nn*** *registers* starting with address ***sss***. **Example:** For x = 34.56, R-CLR will clear **R34 - R89**. **ATTENTION:** For ***nn*** = 0 , clearing will cover the maximum available: • For $sss \in [0; 99]$, it will stop at **R99**. • For $sss \in [100; 111]$, it will stop at **K**. • For $sss \geq 112$, it will stop at the highest local *register* currently allocated.
R-COPY	P.FN R-COPY	[4] (0) {2} Interprets x in the form `sss.nnddd` . Takes ***nn*** *registers* starting with address ***sss*** and copies their contents to ***ddd*** etc. **Example**: For x = 7.03045, R-COPY will copy ***r07***, ***r08***, and ***r09*** into **R45**, **R46**, and **R47**, respectively.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
		For $x < 0$, R-COPY will take ***nn*** *registers* from the backup file instead, starting with *register* number ***|sss|***. Destination will be in *RAM* always. **ATTENTION:** For ***nn*** = 0 , copying will cover the maximum available as explained with R-CLR. Then x must be negative.
R-SORT	P.FN R-SORT	[4] (0) {2} Interprets x in the form `sss.nn`. Sorts the contents of ***nn*** *registers* starting with address ***sss***. **Example**: Assume x = 49.03, $r49$ = 1.2, $r50$ = −3.4, and $r51$ = 0; then R-SORT will return $r49$ = −3.4, $r50$ = 0, and $r51$ = 1.2. **ATTENTION:** For ***nn*** = 0 , sorting will cover the maximum available as explained with R-CLR.
R-SWAP	P.FN R-SWAP	[4] (0) {2} Works like R-COPY but swaps the contents of corresponding source and destination *registers*.[46]
R→D	R→D	[4] (1) {1, 2, 4} → {4} Convert *angles* as described on pp. 161f.
R→MIL	▲ R→MIL	
R→Mπ	R→Mπ	
R↑	R↑	[2,1] Rotates the *stack* contents one level up or down, respectively. See *OMS 1*.
R↓	R↓	
R_∞		Constants (see pp. 113ff).
$R_\odot$		
$R_\oplus$		
s	STAT s	[3] (−2) { } → {2} Takes the statistical data accumulated and returns the *sample SD* s_y and s_x. See *OMS 2* for the output format and pp. 278ff for the formula.

[46] Corresponds to the historic command REGSWAP.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
Sa		Constant (see pp. 113ff).
SAVE	SAVE	[2] (0) Saves your *WP43 state* (program space, *RoV's*, *system flags*, user assignments, and *configuration*, see *OMS 3*) in the backup file SAVFILES/wp43.sav on the *FD* and returns **Saved**. Recall your backup or parts of it using the different flavors of LOAD (cf. p. 55).
SAVEST	I/O SAVEST	[3] (0) Works like SAVE but saves your *WP43 state* in a named file STATE/***name***.s43. On your calculator, you will be prompted by *DMCP* for the file***name*** (choose either **<New File>** [47] or an existing file to overwrite); on your *PC*, a corresponding prompt will appear. Recall this entire *state* or parts of it using the different flavors of LOAD (cf. p. 55)..
SB	BIT SB ***n***	[4+] (1) {10} Sets the specified bit in ***x*** .
Sb		Constant (see pp. 113ff).
SCI	DISP SCI ***n***	[3+] (0) Sets scientific display format (see *OMS 2*); $0 \le n \le 15$ is the number of decimals of the mantissa (ideally, ***n*** should be the number of significant digits minus 1).
scw→kg	U→ m: kg ← short cwt	[5] (1) {2}; {1} → {2} Converts masses. See pp. 150ff.
SDIGS?	INFO SDIGS?	[4] (−1) { } → {1} Returns the number of significant digits set by SETSIG. Also returned by STATUS.
SDL SDR	P.FN P.FN2 SDL ***n*** etc.	[5+] (1) {1, 2} Shifts digits left (right) by ***n*** decimal positions, equivalent to multiplying (dividing) ***x*** by 10^{n}. SDR for *long integers* will end with zero. Compare ASR, SL, and SR for binary *integers*.

[47] You will be in a kind of *AIM*, set to lower case. Use ⬜ to toggle upper case or digits.

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
Se^2		Constant (see pp. 113ff).
SEED	PROB ▲ SEED	[4] (0) {2} Stores a seed ($0 < x \leq 1$) for RAN#. For $x \leq 0$, the seed is taken from the real-time clock; this is a way to generate more random numbers. Cf. RAN#.
SETCHN	DISP ▲ CHINA	[4] (0) Sets regional format preferences. [48]
SETDAT	CLK ▲ SETDAT	[4] (0) {2, 6} Sets the date for the real-time clock. [49]
SETEUR	DISP ▲ EUROPE	[4] (0) Set regional format preferences. [48]
SETIND	DISP ▲ INDIA	
SETJPN	DISP ▲ JAPAN	
SETSIG	MODE SETSIG	[4] (0) {1} Sets the number of significant digits (1 ... 34) for rounding after each operation. SETSIG **0** sets maximum precision. SETSIG is relevant for the precision of the *Solver* among others. See there.
SETTIM	CLK ▲ SETTIM	[4] (0) {2, 5} Sets the time for the real-time clock. [49] Input may be a *real number* formatted `hh.mmss` or a *time*.
SETUK SETUSA	DISP ▲ UK etc.	[4] (0) Set regional format preferences. [48]
Se'^2		Constant (see pp. 113ff).
SF	SF ***n***	[2+] (0) Sets the *flag* specified.
	FLAG SF ***n***	
	MODE SF ***n***	
Sf^{-1}		Constant (see pp. 113ff).

[48] See *OMS 2* about localizing numeric output.

[49] The *Simulator* reads this information from the *PC* clock.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
SHOW	SHOW	[2] (0) {1, 2, 3, 4, 7} Shows all *characters* of x in top numeric row until next keystroke, using small font. For a *real number* up to 34 digits will be displayed, for a *time* up to 35, for a *date* (incl. time) up to 36. For a *complex number*, either part of it will take one *LCD* row if needed. For a *long integer*, up to 298 digits can be shown using up to 7 *LCD* rows; for any greater *integer*, just its most significant 296 digits can be displayed, followed by ellipses. For extracting even greater *integers* use DUMP (p. 37) and see *App. E*. A text string may contain up to 196 *characters*.
sh.t→kg	U→ m: ▲ kg ← short ton	[5] (1) {2}; {1} → {2} Converts masses. See pp. 150ff.
sign	PART sign	[3] (1) {8*}; {1, 2, 10} → {1} Returns 1 for $x > 0$, -1 for $x < 0$, and 0 for $x = 0$ or non-numeric data. Corresponds to the mathematical function $\mathrm{signum}(x)$.
		(1) {3} Returns the unit vector of the *complex number* x (see UNITV). Maintained for backward compatibility only.
SIGNMT	BIT ▼ SIGNMT INTS ▲ SIGNMT	[4] (0) Sets *sign-and-mantissa* mode for operations on *short integers*. See *OMS 2*.
SIM_EQ	MATX SIM EQ *n*	[3+] (0) Opens a window for solving a system of n linear equations $(MATA) \cdot \overrightarrow{MATX} = \overrightarrow{MATB}$. If these matrices and vectors are not defined before, they will be created automatically at execution time. See *OMS 2*.
sin	TRI sin	[2] (1) {2, 3, 8*, 9*}; {1, 4} → {2} Returns the sine of the angle in X (takes x as tagged, else assumes the current *ADM*; see *OMS 2*).

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
sinc	TRI **sinc**	[3] (1) {2, 3, 4, 8*, 9*} Returns $sin(x)/x$ … … for $x \neq 0$ and 1 for $x = 0$. Tagged *DT 4* input will be converted in *radians* before computing. Untagged input is taken as *radians*.
sincπ	TRI **sincπ**	[3] (1) {2, 3, 4, 8*, 9*} Returns $sin(\pi x)/\pi x$ …
sinh	TRI **sinh** EXP **sinh**	[3] (1) {2, 3, 8*, 9*} Returns the hyperbolic sine of x.
SKIP	P.FN P.FN2 **SKIP** ***n***	[5+] (0) Skips ***n*** program steps forwards ($0 \leq n \leq 255$). So, e.g., SKIP **2** skips over the next two steps, going, e.g., from step 123 to step 126. If SKIP attempts to cross an END, an error is thrown. **ATTENTION:** If you edit a program section crossed by one or more BACK, CASE, or SKIP jumps, this may well make manual maintenance of all those individual statements necessary.
SL	BIT ▲ **SL** ***n***	[5+] (1) {10} Shifts all bits of x to the left n (≤ 63) bits.[50] Each bit shifted out of the word is shifted into CARRY, and the previous state of CARRY is lost. New bits generated at the right end of x are all zeros. SL **0** executes as NOP, but loads **L**. See *OMS 2*.
SLVC	ADV **SLVC**	[4] {1, 2, 3} → {1 or 2 or 3} Solves the cubic equation $ax^3 + bx^2 + cx + d = 0$ and tests the result. Its parameters shall be on the input *stack* [***d***, ***c***, ***b***, ***a***, …]. For *real* parameters, SLVC returns the three roots in **X**, **Y**, **Z**, and the discriminant $\Delta = b^2c^2 - 4ac3 - 4b^3d - 27a^2d^2 + 18abcd$ in **T**. Higher *stack registers* are kept unchanged.

[50] SL on your *WP43* works like ***n*** consecutive calls to SL on the *HP-16C*. This applies in analogy to SR as well.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
SLVQ	ADV SLVQ	[3] {1, 2, 3} → {1 or 2 or 3} Solves the quadratic equation $ax^2 + bx + c = 0$ with its parameters on the input *stack* [***c***, ***b***, ***a***, …], and tests the result. The following holds for *real* parameters: • If $\boldsymbol{r} := b^2 - 4ac \geq 0$, SLVQ returns $-\frac{b \pm \sqrt{r}}{2a}$ in $\mathbf{Y}$ and $\mathbf{X}$. In a routine, the step after SLVQ will be executed. • Else it returns the 1st *complex* root in $\mathbf{X}$ and the 2nd in $\mathbf{Y}$ (the *complex conjugate* of the 1st). In a routine, the step after SLVQ will be skipped. In either case (and also for *complex* parameters), SLVQ returns ***r*** in $\mathbf{Z}$. Higher *stack registers* are kept unchanged. $\mathbf{L}$ will contain parameter *c*.
s_m	STAT s_m	[3] (−2) { } → {2} Returns the *standard errors* (i.e. *SDs* of the means $\bar{y}$ and $\bar{x}$). Output format will be like the one of s (see *OMS 2*).
s_{mi}	FIT PLOT s_{mi}	[5] (-1) Pushes the measured precision of the measuring instrument on the *stack* (determined according to the method for *MSA* explained on pp. 295ff and demonstrated in *OMS 2*). Cf. CENTRL on p. 28. Note the basis must be ≥ 30 measured data points or an error will be thrown.
s_{mw}	STAT s_{mw}	[4] (−1) { } → {2} Returns the *standard error* for weighted data, i.e. the *SD* of the mean $\bar{x}_w$.
sn(u,m)	X.FN Ellipt sn	[4] (2) {2, 3} Returns the *elliptic sine* of the argument ***u*** = ***x*** and the parameter ***m*** = ***y***. See p. 317.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
SNAP	SNAP	[2] (0) Stores a snapshot of the calculator screen (a.k.a. screenshot) in a `.bmp` file in the directory `SCREENS` on the *FD*. The file (its name comprises date and time of storage, see *OMS 3*) can be transferred to your *PC* and dealt with there. Entered in *PEM*, a step SNAP is inserted. If you want to store a snapshot in *PEM* instead, press 🔵 + E simultaneously (this will work also if you want to take a snapshot of a *temporary message*).
SOLVE	ADV SOLVE ***label*** (called in *RUM*)	[3+] {2, 3} Solves the equation $f(v) = 0$, with f as specified by the routine ***label***. See common remarks below.
	ADV SOLVE ***v*** (called in *PEM*)	[3+] {2, 3} Solves the equation $f(v) = 0$, with f as specified by PGMSLV. Additionally acts as binary test, so the next program step will be skipped if SOLVE fails to find a root. **ATTENTION:** SOLVE fills the *stack* with x before calling the routine specified.
		Common remarks: Two initial guesses of the root may be supplied in **X** and **Y** when calling SOLVE. It returns… • v_{root} in **X** with accuracy determined by SETSIG, • the penultimate ***v***-value tested in **Y**, • $f(v_{root})$ in **Z**, and • **T** contains … **0** if SOLVE found a root. **1** if it found a sign reversal but $f(v)$ is not 'small'. **2** if it found a local or asymptotic extremum. **3** if ≥ 1 start value is out of function domain. **4** if SOLVE found $f(v)$ looking constant. Higher *stack registers* are not affected. See pp. 325f and *OMS 4* also for examples. – SOLVE may be aborted by EXIT.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
Solver	EQN Solver	*Submenu* for solving a given equation. See *OMS 4*.
SOLVING		*System flags* (see pp. 127ff)
SPCRES		
SPEC?	TEST ▲ SPEC?	[5] (0) True if x is 'special' (i.e. ±∞ or NaN).
SQRT	CAT. FCNS SQRT	Points to $\sqrt{x}$ on p. 109. Just for backward compatibility.
SR	BIT ▲ SR *n*	[5+] (1) {10} Shifts all bits in x to the right n (≤ 63) bits in analogy to SL. SR **0** executes as NOP, but loads **L**. Compare ASR on p. 23 and see *OMS 2*.
SSIZE8		*System flag* (see pp. 127ff)
SSIZE?	INFO SSIZE?	[3] (−1) { } → {1} Returns the number of *stack registers* currently allocated, 4 or 8.
STAT	STAT	*Menu*. See p. 145.
STATS	Reserved *variable*	for the statistics matrix (see *OMS 2*).
STATUS	FLAG STATUS	[3] Displays the number of free bytes in *RAM*, all *flags* set, number of local *registers* allocated, and *system variables*. See *OMS 5*.
STK	STK	*Menu*. See p. 145.
STO	STO *r*	[1+] (0) Stores a copy of x into the *RoV* specified.[51]
STOCFG	STO Config *r*	[3+] (0) Stores the current calculator *configuration* in an *RoV* for later use as described in *OMS 2* and *6*. RCLCFG recalls such data. Cf. also RESET on p. 75.[51]
STOEL	MATX STOEL STO ...EL	[3] (1) {1, 2, 3} Stores a copy of x into the current element a_{ij} of the *indexed matrix*. Cf. RCLEL.[51]
STOIJ	MATX ▲ STOIJ STO ...IJ	[3] (1) {1} Sets the index pointers to IP(x) (column number) and IP(y) (row number). Cf. RCLIJ.[51]

[51] **L** will not be loaded. This is sensible since x stays unchanged.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
STOMAX	STO max *r*	[2+] (0) {1, 2, 4, 5, 6, 10} Stores the maximum (or minimum) of ***r*** and ***x*** into the *RoV* specified.[52]
	STO ▲ *r*	
STOMIN	STO min *r*	
	STO ▼ *r*	
STOP	R/S	[1] (0) Stops program execution. May be inserted in a program to wait for input, for instance.
STOSTK	STO Stack *r*	[3+] (0) Stores the entire *stack* in a set of 4 or 8 *registers*, starting at the destination address specified. Cf. RCLSTK.[51]
STO+	STO + *r*	[2+] (0) Executes the specified operation on ***r*** and stores the result (e.g., $r - x$) in the *RoV* specified.[52] Note every *RoV* can be a target, so STO× **X** is legal as well.
STO−	STO − *r*	
STO×	STO × *r*	
STO/	STO / *r*	
sto→kg	U→ m: kg ← stone	[5] (1) {2}; {1} → {2} Converts masses. See pp. 150ff.
STRI?	TEST STRI?	[4] (0) Returns true if ***x*** is a *text string*.[53]
STRING	CATALOG VARS STRING	*Submenu* of all *text string variables* defined at execution time. See pp. 133f.
SUM	STAT SUM	[3] (−2) { } → {2} Recalls the linear sums *Σy* and *Σx*. Useful in basic *2D* vector algebra. Output labeled in analogy to s.
s_w	STAT s_w	[4] (−1) { } → {2} Returns the sample *SD* for weighted data (where the weight ***y*** of each data point ***x*** was entered via Σ+). See pp. 269ff for the formula.

[52] Only legal operations according to the *DT* matrices in *OMS 2* will work. See also the examples given there. **L** will not be loaded.

[53] This works almost like STR? in *HP-42S*.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
s_{xy}	FIT s_{xy}	[4] (−1) { } → {2} Returns the *sample covariance* for the two data sets {*x*} and {*y*} entered via Σ+. See pp. 278ff for the formula and COV for the *population covariance*.
SYS.FL		*Submenu* of all *system flags* accessible by CF, FC?, FF, FS?, SF, etc.
s(a)	FIT ▲ s(a)	[4] (−2) { } → {2} Returns the standard errors s(a_1) (in **Y**), and s(a_0) (in **X**) for the parameters of the line fitted through the data points accumulated in STATS following the fit model chosen. Works for EXPF, LINF, LOGF, ORTHOF, and POWERF only – not for CAUCHF, GAUSSF, HYPF, PARABF, and ROOTF. Cf. CENTRL (p. 28), s_{mi} (p. 85), and see pp. 286ff.
S.INTS	CATALOG VARS S.INTS	*Submenu* of all *short integer variables* defined at execution time. See pp. 133f.
s→day s→year	U→ Time: s → day etc.	[4] (1) {2}; {1} → {2} Convert times. See pp. 150ff.
T	Reserved variable for register **T**.	
T_0		Constant (see pp. 113ff).
tan	TRI tan	[2] (1) {2, 3, 8*, 9*}; {1, 4} → {2} Returns the tangent of the angle in **X** (takes ***x*** as tagged, else assumes the current *ADM*; see *OMS 2*). May return "*Not a Number*" for ***x*** = ±90° or equivalents if SPCRES is set.
tanh	EXP tanh TRI tanh	[3] (1) {2, 3, 8*, 9*} Returns the hyperbolic tangent of ***x***.
tb_I→mℓ	U→ V: ▲ ▲ mℓ ← table spoon$_I$	[6,5] (1) {2}; {1} → {2} Convert kitchen volumes. See pp. 150ff.
tb_{UK}→mℓ	U→ V: ▲ mℓ ← tbsp$_{UK}$	
Tb_{US}→mℓ	U→ V: ▼ mℓ ← Tbsp$_{US}$	

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
TDISP	CLK ▲ TDISP ***n***	[4+] (0) Sets time display format. TDISP 0 allows for the full string from *hours* to *milliseconds* (trailing decimal zeros will not be shown), TDISP 1 or 2 truncates it to *hours* : *minutes* only, TDISP 3 to *hours* : *minutes* : *seconds*, and 4 ≤ ***n*** ≤ 6 displays also ***n*** − 3 digits for decimal fractions of *seconds*. **Example:** With TDISP **5**, a time string might look like 157:26:38.49
TDM24		*System flag* (see pp. 127ff)
TEST	TEST	*Menu*. See p. 145.
TICKS	P.FN TICKS	[4] (−1) { } → {1} Returns the number of ticks from the real-time clock at execution time. 1 *tick* = 0.1 s. Counting starts when the calculator is turned on. You can use TICKS for measuring execution times (see an example in *OMS 4*).
TIME	CLK TIME	[3] (−1) { } → {5} Recalls the time from the real-time clock (or the *PC* clock for the *Simulator*) at execution. See *OMS 2* for the output format.
TIMER	⏱	[2] Opens the timer (a.k.a. stopwatch) application based on the calculator real-time clock. The time counted can be indicated up to 99:59:59.9 (i.e. almost 100 hours – care for reliable power supply), thereafter an overflow will occur but the timer will continue. See 0.1s, ADD, T.RES, and in particular *OMS 5* for more information.
TIMES	CATALOG VARS TIMES	*Submenu* of all *time variables* defined at execution time. See pp. 133f.
Time:	U→ Time:	*Submenu* for time conversions. See pp. 150ff.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
T_n	X.FN Orthog T_n	[4] (2) {2}; {1} → {2} *Čebyšëv polynomials of 1st kind*. See pp. 286f.
TONE	I/O TONE ***n***	[4+] (0) Sounds a tone according to ***n*** (= 0 … 9) .
ton→kg	U→ m: ▲ kg ← ton	[5] (1) {2}; {1} → {2} Converts masses. See pp. 150ff.
TOP?	TEST TOP?	[4] (0) Returns ... • **false** if called with the *PP* in a subroutine; • **true** if called in the top routine (i.e. if the program-running *flag* is set and the *SRS* pointer is clear).
torr→Pa	U→ F&p: Pa ← torr	[5] (1) {2}; {1} → {2} Converts pressures. See pp. 150ff.
T_{PL}		Constants (see pp. 113ff).
t_{PL}		
$t_p(x)$	PROB t: $t_p(x)$ etc.	[4] (1) {2}; {1} → {2} *Student's t distribution*. The degrees of freedom are stored in **I**. $t_{▲}(x)$ equals Q(t) on *HP-21S*. See *OMS 2* for an application example and pp. 269ff.
$t_{▲}(x)$		
$t_{▲}(x)$		
$t^{-1}(p)$		
TRACE	SF T	*System flag* (see pp. 127ff)
TRANS	CAT. FCNS TRANS	Equals $[M]^T$ (p. 108). For backward compatibility only.
TRI	TRI	*Menu*. See p. 145.
troz→g	U→ m: kg ← troy oz. etc.	[5] (1) {2}; {1} → {2} Convert masses and volumes. See pp. 150ff.
trz→kg		
ts_I→mℓ	U→ V: ▲ ▲ mℓ ← tea $spoon_I$	
ts_{UK}→mℓ	U→ V: ▲ mℓ ← tsp_{UK}	
ts_{US}→mℓ	U→ V: ▼ mℓ ← tsp_{US}	
TVM	FIN TVM	*Submenu* for the *Time Value of Money* (see *OMS 5*).

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
T.RES	⏱ RESET	[3] (0) Resets the timer to zero without changing its status (running or stopped). Deletes total time if applicable.[54]
t:	PROB t:	*Submenu* for the *t distribution*. See p. 144.
t⇄	STK t⇄ *r*	[3] Swaps ***t*** and ***r***, in analogy to $x\rightleftarrows$.
ULP?	INFO ULP?	[4] (1) {1, 2} Returns 1× the smallest power of ten which can be added to x or subtracted from x to actually change the (internal) value of x in your *WP43* in the mode set. Thus, 1 is returned for *integers*. Indicated in STATUS.
U_n	X.FN Orthog U_n	[4] (2) {2}; {1} → {2} *Čebyšëv polynomials of 2nd kind*. See pp. 286f.
UNDO	↶	[2] Recalls the *stack*, STATS, summation *registers*, and all *system flags* as they were before the last operation executed – unless that operation is printed in red or orange in this *IOI*.
UNITV	MATX UNITV	[4] (1) {8, 9} Returns the unit vector for the matrix x. Each element of the matrix is adjusted so its overall Euclidean norm becomes 1 (cf. ENORM on p. 39); for a vector, its *magnitude* will become 1.
	CPX UNITV	[3] (1) {3} Returns a *complex number* with *magnitude* $\lvert r\rvert = 1$ in direction of x.
UNSIGN	BIT ▼ UNSIGN INTS ▲ UNSIGN	[4] (0) Sets *unsigned* mode for operations on *short integers* (≈ UNSGN on *HP-16C*). See *OMS 2*.
USB		*System flags* (see pp. 127ff)
USER	USER	
U→	U→	*Menu*. See p. 145.

[54] Note this RESET is just a *label* in TIMER, not the command RESET above.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
VAR		*Submenu* of all *variables* defined at execution time when calling STO, RCL, etc.
VARMNU	P.FN VarMNU ***label***	[3+] Creates a *variable menu* (i.e. a *menu* of *variables*) using the MVAR steps following the *global label* specified here. See *OMS 3.*
VARS	CATALOG VARS	*Submenu* of all *variables* defined at execution time. See pp. 133f.
VERS?	INFO VERS?	[4] (0) Shows firmware version and build number until next keystroke (see *OMS 2*).
VIEW	VIEW ***s***	[2+] (0) Displays ***s*** left-adjusted in the top numeric row (and below if it needs more space) until the next key is pressed (in programs until the next VIEW or AVIEW occurs). Cf. AVIEW. **Example:** If a *variable* ***Test12*** contains **−123.45** then VIEW α Test12 ENTER↑ will display in top row Test12 = −123.45
VMDISP		*System flag* (see pp. 127ff)
V_m		Constant (see pp. 113ff).
VOL	CATALOG FCNS VOL ***n***	[4+] (0) Sets the buzzer volume to level ***n*** (1…11).
VOL+	I/O VOL+	[4] (0) Increases the buzzer volume by one level.
VOL-	I/O VOL-	[4] (0) Decreases the buzzer volume by one level.
VOL?	INFO ▼ VOL?	[4] (-1) Returns the present buzzer volume.
V.ALL	CAT. VARS ALL	*Submenu* of all *variables* defined at execution time. See pp. 133f.
V:	U→ V:	*Submenu* for conversions of volumes. S. p. 150.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
V∢	[∠]	[2] (2) {8} → {4} Returns the *angle* between two real *2D* or *3D* vectors $$\vartheta = arccos\left(\frac{\vec{v}_1 \cdot \vec{v}_2}{|\vec{v}_1| \cdot |\vec{v}_2|}\right)$$
WDAY	[CLK] WDAY	[3] (1) {2, 6} → {1} Assumes x containing a *date* in the format selected (or a *real number* in a corresponding format) and returns the name of the respective day and a corresponding *integer* (Monday = 1).[55]
$Weibl_p$ $Weibl_{\blacktriangle}$ $Weibl_{\blacktriangle}$ $Weibl^{-1}$	[PROB] [▮] Weibl: $Weibl_p$ etc.	[5] (1) {2}; {1} → {2} *Weibull distribution* with its shape parameter ***b*** in $\mathbf{I}$ and its characteristic lifetime ***T*** in $\mathbf{J}$. See pp. 269ff. $Weibl^{-1}$ returns the survival time t_s for a given probability ***p*** in $\mathbf{X}$, with ***b*** in $\mathbf{I}$ and ***T*** in $\mathbf{J}$.
Weibl:	[PROB] [▮] Weibl:	*Submenu*. See p. 144.
WHO?	[INFO] WHO?	[3] (0) Displays credits to the brave men who made this project work (temporary message).
Wh→J	[U→] E: J ← Wh	[4] (1) {2}; {1} → {2} Converts energies. See pp. 150ff.
W_m W_p W^{-1}	[X.FN] [▲] W_m etc.	[4] (1) {2, 3}; {1} → {2} W_p returns the principal branch of *Lambert's* W for given $x \geq -1/e$. W_m returns its negative branch (works for $x \in \mathbb{R}$ only). W^{-1} returns x for a given W_p (≥ −1). See pp. 316ff.
WRITEP	[I/O] WRITEP	[3] (0) Writes the *current program* on the *FD* (see *App. E* for the *Simulator*). You will be prompted for a filename. A file carrying an identical name will be overwritten. Cf. READP and *OMS 3*.

[55] *TN*: These day numbers correspond to Chinese weekdays 1 to 6 directly. For Portuguese and Brazilian weekdays ('segunda-feira' etc.), add 1 to days 1 to 5.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
WSIZE	BIT ▼ WSIZE ***n*** INTS ▲ WSIZE ***n***	[4+] (0) Sets the *word size* (a.k.a. bit width, $1 \le n \le 64$) for *DT 10*.[56] WSIZE 0 sets the *word* size to maximum, i.e. 64 *bits*.[57] Reducing *word size* truncates *short integer* values in **L** and in the *stack registers*. All other memory content stays as is (see *App. B* on pp. 197f). Increasing the *word size* adds significant empty bits to *short integer* values in **L** and in the *stack registers*. All other memory content stays as is.
WSIZE?	INFO WSIZE?	[4] (−1) { } → {1} Recalls the *word* size set.
W→hp$_E$ W→hp$_M$ W→hp$_{UK}$	U→ P: W → hp$_E$ etc.	[4] (1) {2}; {1} → {2} Convert powers. See pp. 150ff.
X	Reserved *variable* for *stack register* **X**.	
$\hat{x}$	FIT $\hat{x}$	[3] (1) {2}; {1} → {2} Returns a forecast ***x*** for a given ***y*** (in **X**) according to the curve fit model chosen (note it may return 2 results for 3-parameter models). Cf. L.R. on p. 57.
$\bar{x}$	STAT $\bar{x}$	[3] (−2) { } → {2} Calculates the *arithmetic means* of the ***y***- and ***x***-data accumulated and pushes these means on the *stack*. See also s, s_m, and σ.
x^2	EXP x^2	[2] (1) {1, 2, 3, 8*, 9*, 10} Return the square or cube of ***x***, respectively.
x^3	EXP x^3	
XEQ	XEQ ***label*** or ***funct.name***	[1+] (0) Executes the function specified or the routine with the *label* specified. In *PEM*, XEQ inserts a call to the subroutine with the *label* specified.

[56] Works almost like on *HP-16C*, but with the parameter trailing the command instead of taken from **X** there.

[57] WSIZE works as it does on *WP 34S*.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
XEQ.SKP	P.FN P.FN2 XEQ.SKP	[5] (0) Works like XEQ but will return from the subroutine called to the step two steps behind the XEQ.SKP instruction. Cf. RTN.SKP.
$\bar{x}_G$	STAT $\bar{x}_G$	[4] (−2) { } → {2} Calculates the *geometric means* of the y- and x-data accumulated and pushes them on the *stack*. See pp. 278ff for the formula. Output format will be similar to the one of $\bar{x}$. See also ε, ε_m, and ε_p on pp. 101f).
$\bar{x}_H$	STAT ▲ $\bar{x}_H$	[5] (−2) { } → {2} Calculates the *harmonic means* of the y- and x-data accumulated and pushes them on the *stack*.
x_{max}	STAT ▲ x_{max} etc.	[5] (−2) { } → {2} Returns the maximum (or minimum) of the y- and x-data accumulated and pushes them on the *stack*.
x_{min}		
XNOR	BIT XNOR	[4, 3] (2) Boole's operations working in analogy to
XOR	BIT XOR	AND. Cf. p. 22.
$\bar{x}_{RMS}$	STAT ▲ $\bar{x}_{RMS}$	[5] (−2) { } → {2} Calculates the *quadratic means* of the y- and x-data accumulated and pushes them on the *stack*. Quadratic means are often used in electrical engineering with alternating currents.
$\bar{x}_w$	STAT $\bar{x}_w$	[4] (−1) { } → {2} Returns the *arithmetic mean* for weighted data (where the weight y of each data point x was entered via Σ+). Look up pp. 278ff for the formula. See also s_w and s_{mw}.
$\sqrt[x]{y}$	EXP $\sqrt[x]{y}$	[3] (2) Returns the xth root of y. For $y < 0$ and CPXRES set, $\sqrt[x]{y}$ may return *complex numbers*.
X.FN	X.FN	*Menu*. See p. 146.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
x!	x!	[2] (1) {1, 10} Returns the *factorial* ***n!***. Note this is only defined for positive *integers*. *20!* is the biggest factorial < 2^{64}. *165!* returns the biggest *long integer* result you can display entirely using SHOW. *450!* is the maximum for *long integers* (returning a 1001-digit *integer*). Inputs > 450 will be converted to *reals* before calculating ***x!***
		(1) {2, 3} Returns *Γ(x + 1).* *2 123.549 956 662 463 236 31* is the maximum *real* input (returning almost 10^{6145}).
x:	U→ **x:**	*Submenu* for conversions of distances. See p. 150.
x→DATE	CLK **x→DATE**	[3] (1) {2} → {6} Takes the *real number* ***x*** as a date coded in the date format selected (YMD, DMY, or MDY) and converts it to a proper *date*.
x→α	α.FN **x→α** ***r***	[3+] (1) {1, 2, 10} → {7} Interprets IP(***x***) as a *character code*, converts it to the respective *character*, and appends it to ***r***. For destination **X**, it returns just the *character* converted. If **X** contains a text string already, **x→α** appends the entire string to ***r***.[58]
x⇄	x⇄ ***r*** STK **x⇄** ***r***	[2+] Swaps ***x*** and ***r***, in analogy to ***x⇄y***. Will be listed like **x⇄ J**, **x⇄ .12**, **x⇄ →12**, etc. in programs.
x⇄y	x⇄y	[1] Swaps the contents of *stack registers* **X** and **Y**.

[58] This works like XTOA on *HP-42S* but with an arbitrary destination instead of the α-*register.*

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
x=+0? x=-0?	TEST ▲ x=+0? etc.	[4] (0) {1, 2, 3, 10} These two tests are for comparing *short integers* in modes 1COMPL and SIGNMT, and for *real* or *complex numbers* if SPCRES is set. Then, e.g., 0/(−7) will display −0.
x<? x≤?	TEST x<? s etc.	[3+] (0) {1, 2, 4, 5, 6, 7, 10} Compare x with s. *Text strings* are compared according to their sorting order. See also *Addressing Tables* in *OMS 1*. **Example:** TEST x<? K compares x with k, and will be listed as x<? K in a routine. It will return true if $x < k$ at execution time.
x=?	TEST x=? s	[3+] (0) Compares x with s. Cf. x<? for more.
x≈?	TEST ▲ x≈? s	[4+] (0) {2, 3, 4, 5, 8, 9} Will be true if the rounded values of x and s are equal (cf. ROUND on p. 78). Cf. x<? for more.
x≠?	TEST x≠? s	[3+] (0) Compares x with s.
x≥?	TEST x≥? s	[3+] (0) {1, 2, 4, 5, 6, 7, 10} Compare x with s.
x>?	etc.	Cf. x<? for more.
Y	Reserved *variable* for *stack register* Y.	
ŷ	FIT ŷ	[3] (1) {2}; {1} → {2} Returns a forecast y (in X) for a given x according to the curve fit model chosen. Cf. L.R. on p. 57.
yd.→m	U→ x: m ← yard	[5] (1) {2}; {1} → {2} Converts distances. See pp. 150ff.
YEAR	CLK YEAR	[4] (1) {2, 6} → {1} Assumes x containing a *date* in the format selected (or a *real number* in corresponding format) and extracts the year.
year→s	U→ Time: s ← year	[4] (1) {2}; {1} → {2} Convert times and distances. See pp. 150ff.
yĭn→m	U→ x: ▲ m ← yĭn	

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
YMD		*System flag* (see pp. 127ff)
y^x	EXP y^x	[2] (2) Raises y to the power of x. For $y < 0$ and CPXRES set, it may return *complex numbers*.
$Y_y(x)$	X.FN $Y_y(x)$	[4] (2) {2}; {1} → {2} Returns the *Bessel function of second kind* and order y. See p. 319.
Y.MD	CLK ▲ Y.MD	[4] (0) Sets the format `yyyy-mm-dd` for *dates*.
$y\rightleftarrows$	STK $y\rightleftarrows$ r	[3+] Swaps y and r, in analogy to $x\rightleftarrows$.
Z	Reserved *variable* for *stack register* Z.	
Z_0		Constant (see pp. 113ff).
zhàng>m	U→ x: ▲ m ← zhàng	[5] (1) {2}; {1} → {2} Converts distances. See pp. 150ff.
ZOOM	EQN Solver Draw ZOOM FIT ■ ASSESS ZOOM	[5] (0) Zooms the current plot out. See *OMS 4.*
$z\rightleftarrows$	STK $z\rightleftarrows$ r	[3+] Swaps z and r, in analogy to $x\rightleftarrows$.
α		Constant (see pp. 113ff).
αCAP		*System flag* (see pp. 127ff)
αINTL	CATALOG CHARS αINTL	*Submenu.* See pp. 133f.
	■ 1/x (A)	*Catalog* in *AIM* (see p. 147).
αLENG?	INFO αLENG? s	[4+] (−1) { } → {1} Returns the number of *characters* found in the *text string* s.[59]
	α.FN αLENG? s	
αMATH	CAT. CHARS αMATH	*Submenu.* See pp. 133ff.
	■ − (≈)	*Menu* in *AIM*. See p. 148.

[59] This will throw an error if there is no *text string* or an empty string in r at execution time. It corresponds to ALENG, POSA, ASHF, or ATOX on *HP-42S*, respectively

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
αPOS?	INFO αPOS? *s* α.FN αPOS? *s*	[4+] (−1) { } → {1} Looks in the *text string* ***s*** for the *target* given in **X**. If a match is found, αPOS returns the position number where the *target* was found (counting the left-most *character* as position 0). If a match is not found, αPOS returns −1. [59] The *target* may be an individual *character code* or a *text string*. αPOS saves a copy of the *target* in **L**.
αRL	α.FN αRL *r*	[3+] (0) Rotates the *text string* ***r*** by ***x*** (≥ *0*) symbols to the left. αRL **0** executes as NOP, but loads **L**. [59]
αRR	α.FN αRR *r*	[3+] (0) Works like αRL but rotates to the right.
αSL	α.FN αSL *r*	[3+] (0) Shifts the ***x*** leftmost symbols out of the *text string* ***r***. This allows for deleting the first ***x*** symbols in a string. αSL 0 executes as NOP, but loads **L**. [59]
αSR	α.FN αSR *r*	[3+] (0) Works like αSL but for the ***x*** rightmost symbols out of ***r***, deleting the last ***x*** symbols in the string. [59]
α.FN	α.FN	*Menu*. See p. 148.
A...Ω	CAT. CHARS A...Ω	*Submenu* of Greek letters, see pp. 133ff.
α·	CAT. CHARS α·	*Submenu* for punctuation, see pp. 133ff.
	▣ .	*Menu* in *AIM*. See p. 148.
α→x	α.FN α→x *r*	[3+] (−1) { } → {10_{16}} Pushes the *character code* of the leftmost *character* in the *text string* ***r*** on the *stack* and removes this *character* from the string [59]
β(x,y)	X.FN ▲ β(x,y)	[4] (2) {2, 3}; {1} → {2} Returns *Euler's Beta function* $B(x,y)=\frac{\Gamma(x)\,\Gamma(y)}{\Gamma(x+y)}$ with $Re(x)>0$ and $Re(y)>0$. Called ***β*** here to avoid ambiguity. See $\Gamma(x)$.

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
γ		Constants (see pp. 113ff).
γ_{EM}		
γ_p		
Γ_p	X.FN ▲ Γ_p	[4,5] (2) {1, 2} Returns the *regularized gamma function* (one of two kinds; see pp. 316ff).
Γ_q	X.FN ▲ Γ_q	
Γ_{xy}	X.FN ▲ Γ_{xy}	[5] (2) {2}; {1} → {2} Returns the *upper* or *lower incomplete Gamma function*. See pp. 316ff.
γ_{xy}	X.FN ▲ γ_{xy}	
Γ(x)	PROB ▲ Γ(x)	[4] (1) {2, 3}; {1} → {2} Returns $\Gamma(x)$. Note x! calls $\Gamma(x+1)$. Cf. also LNΓ on p. 55.
Δx	STAT ▲ Δx	[5] (−2) { } → {2} Returns the range (i.e. ***max*** – ***min***) of the ***y***- and ***x***-data accumulated in STATS and pushes them on the *stack*.
δx	CATALOG PROGS δx	Reserved predefined *global label* to set the step size for f'(x) and f"(x) – see *OMS 4*.
$\Delta\nu_{Cs}$		Constant (see pp. 113ff).
Δ%	Δ%	[2] (1) {2}; {1} → {2} Returns $100\ {}^{x-y}/_{y}$ leaving ***y*** unchanged. You can use it also for calculating markups or margins as explained in *OMS 2*.
ε	STAT ε	[4] (−2) { } → {2} Calculates the *scattering factors* ε_y and ε_x for *log-normally* distributed sample data and pushes them on the *stack*. ε works for the *geometric mean* $\bar{x}_g$ in analogy to the *SD* for the *arithmetic mean* $\bar{x}$ but multiplicative instead of additive. See pp. 269ff.
ε_0		Constant (see pp. 113ff).

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
ε_m	STAT ε_m	[4] (−2) { } → {2} Works like ε but returns the *scattering factors* of the two *geometric means* (in analogy to the *standard error* for *arithmetic means*).
ε_p	STAT ε_p	[4] (−2) { } → {2} Works like ε but returns the *scattering factors* of the two populations.
ζ(x)	X.FN ▲ ζ(x)	[5] (1) {2, 3} Returns *Riemann's Zeta*. See p. 322.
Z(φ,m)	X.FN Ellipt Z(φ,m)	[5] (1) {2, 3} Returns *Jacobi's Zeta*. See p. 317. Note the name of this function begins with a capital Greek zeta (not with a Latin Z, watch the sorting order).
λ_C		Constants (see pp. 113ff).
λ_{Cn}		
λ_{Cp}		
μ_0		Constants (see pp. 113ff).
μ_B		
μ_e		
μ_e/μ_B		
μ_n		
μ_p		
μ_u		
μ_μ		
π	π	[2] (−1) { } → {2} Recalls π. May be used in *complex* input, too.
Π_n	ADV Π_n ***label***	[3+] Computes a finite product using the routine specified. See *OMS 4*. **ATTENTION:** Π_n fills all *stack registers* with ***x*** before calling the routine specified.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
Π(n,m)	X.FN Ellipt Π(n,m)	[4] (2) {2, 3} Returns the *complete elliptic integral of the 3rd kind* for ***n*** = ***x*** and *real number* ***m*** = ***y***. Note 0 ≤ ***m*** < 1. See p. 317.
Σ	Σ	*Menu*. See p. 149.
σ	STAT σ	[3] (−2) { } → {2} Works like s but returns the *SD* of the two *populations* instead. See pp. 269ff.
σ_B		Constant (see pp. 113ff).
$\Sigma 1/x$ $\Sigma 1/x^2$ $\Sigma 1/y$ $\Sigma 1/y^2$	Σ ▲ $\Sigma 1/x$ etc.	[4] (−1) { } → {2} Recall the corresponding statistical sums, necessary for statistical analyses and regressions beyond the linear model. Calling these sums by names significantly improves program readability. Note they are stored in dedicated summation *registers* of your *WP43* (see *OMS 1* and *App. B* on pp. 187ff.).
$\Sigma \ln^2 x$ $\Sigma \ln^2 y$ Σlnx Σlnxy Σlny	Σ $\Sigma \ln^2 x$ etc.	For space reasons, two labels are abbreviated: **Σlnxy** denotes $\sum ln(x)ln(y)$ and **Σlny/x** denotes $\sum ln(y)/x$. Note that logarithmic or inverted sums may become infinite or even non-numeric depending on your input data. If this happens no error will be thrown, regardless of the status of SPCRES. Do not bother.
Σlny/x	Σ Σlny/x	
Σ_n	ADV Σ_n ***label***	[3+] Computes a finite sum using the routine specified. See *OMS 4*. **ATTENTION:** Σ_n fills all *stack registers* with ***x*** before calling the routine specified.
σ_w	STAT σ_w	[4] (−1) { } → {2} Works like s_w but returns the *SD* of the *population* instead. See pp. 269ff.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
Σx **Σx²**	Σ **Σx** etc.	[3,4,5] (−1) { } → {2} Recall the corresponding statistical sums, necessary for statistical analyses and regressions (see **Σ 1/x** for more).
Σx²lny **Σx²y**	Σ **Σx²lny** etc.	
Σx²/y	Σ ▲ **Σx²/y**	
Σx³ **Σx⁴**	Σ ▲ **Σx³** etc.	
Σxlny	Σ **Σxlny**	
Σxy	Σ **Σxy**	
Σx/y	Σ ▲ **Σx/y**	
Σy **Σy²**	Σ **Σy** etc.	
Σylnx	Σ **Σylnx**	
Σ+	FIT **Σ+** [60]	[3] {8} → {2} If **X** contains an ***n*** × 2 matrix then Σ+ copies it to STATS. The *LCD* will show the last data point added (equivalent to the last row of the input matrix) and said matrix will be saved in **L**. See *OMS 2*.
	STAT **Σ+**	{1, 2} Adds one *2D* data point, i.e. adds one row to STATS. Note both **X** and **Y** must contain *DT 1* or *2*.
(Σ+)	in TIMER	Cf. ADD.
Σ-	FIT **Σ-** [60]	[3] (-2) {1, 2} Reverts Σ+, deleting the last row of STATS and returning this last data point to the *stack*.
	STAT **Σ-**	
Φ		Constants (see pp. 113ff).
Φ_0		

[60] Σ+ and Σ− return *temporary information* (as shown in *OMS 2*) and disable *ASL*.

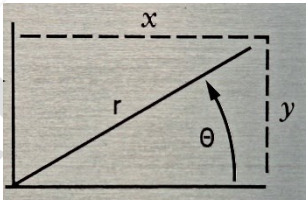
DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
$\chi^2_P(x)$ $\chi^2_{\blacktriangle}(x)$ $\chi^2_{\blacktriangle}(x)$ $(\chi^2)^{-1}$	PROB χ^2: $\chi^2_P(x)$ etc.	[4] (1) {2}; {1} → {2} *Chi-square distribution* (with its degrees of freedom given in **I**). $\chi^2_{\blacktriangle}(x)$ equals $Q(\chi^2)$ on *HP-21S*. See *OMS 2* for an application example and pp. 269ff.
χ^2:	PROB χ^2:	*Submenu*. See p. 144.
$\psi(u,m)$	X.FN Ellipt $\psi(u,m)$	[5] (2) {2, 3} Returns the *elliptic amplitude* for the argument $\boldsymbol{u} = \boldsymbol{x}$ and parameter $\boldsymbol{m} = \boldsymbol{y}$. See p. 317.
ω		Constant (see pp. 113ff).
$(-1)^x$	X.FN ▲ $(-1)^x$	[5] (1) {1, 2, 3, 8*, 9*, 10} If $\boldsymbol{x}$ is non-integer, returns $cos(\pi\,\boldsymbol{x}) + i\,sin(\pi\,\boldsymbol{x})$.
$[M]^T$	MATX $[M]^T$	[3] (1) {8, 9} Returns the transpose of the matrix $\boldsymbol{x}$. The transpose is another matrix with rows changed by columns. If $\boldsymbol{A}$ is an $\boldsymbol{n}\times\boldsymbol{m}$ matrix and $\boldsymbol{a_{ij}}$ is an element of it then $\boldsymbol{A^T}$ will be an $\boldsymbol{m}\times\boldsymbol{n}$ matrix $\boldsymbol{B}$ with $\boldsymbol{b_{ij}} = \boldsymbol{a_{ji}}$. The transpose is done in-situ and does not require any additional memory.
$[M]^{-1}$	MATX $[M]^{-1}$	[3] (0) {8, 9} Takes the square matrix in **X** and inverts it in-situ.
+	+	[1] (2) Returns $\boldsymbol{y} + \boldsymbol{x}$ for compatible objects. [61]
+/-	+/− for closed $\boldsymbol{x}$	[1] (1) 'Unary minus', returns $\boldsymbol{x} \times (-1)$.[61]
±∞?	INFO ■ ±∞?	[4] (0) {2} → {1} Tests $\boldsymbol{x}$ for infinity. Returns true 1 for $\boldsymbol{x} = +\infty$, true −1 for $\boldsymbol{x} = -\infty$, and false 0 else.

[61] See the *DT* combination tables in *OMS 2* for details and compatibility. Note that sizes must match for matrix operations; and for dividing by a matrix $\boldsymbol{x}$, it must be invertible.

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
-	[−]	[1] (2) Returns $y-x$ for compatible numeric objects.[61]
-∞		Constant (see pp. 113ff).
×	[×]	[1] (2) Returns $y \times x$ for compatible numeric objects.[61]
×MOD	INTS ×MOD	[4] (3) {1, 2, 10} Returns $(z \times y) \bmod x$ for $x > 1, y > 0, z > 0$.[62]
/	[/]	[1] (2) Like ×, but returns $y \times x^{-1}$.[61] Returns $y \operatorname{div} x$ if both y and x are of *DT 1* or *10*; cf. IDIV.
^MOD	INTS ^MOD	[4] (3) {1, 10} Returns $(z^{y}) \bmod x$ for $x > 1$, $y > 0$, $z > 0$.[62]
→		Reserved symbol for indirect addressing.
→BIN	INTS [▼] →BIN	[4] (1) {1, 2} → {1} Converts a closed number to a binary *short integer*.
→DATE	CLK →DATE	[3] (3) {1, 2} → {6} Assumes three components of a date (year, month, and day) supplied on the *stack* in proper order for the date format selected and converts them to a single *date* in x. Thus inverts DATE→.
→DEC	INTS [▼] →DEC	[4] (1) {1, 2} → {1} Converts a closed number to a decimal *short integer*.
→DEG	[∡→] →DEG	[3] (1) {1, 2, 4} → {4} Convert *angles* as described on pp. 161f.
→D.MS	[∡→] →D.MS	
→GRAD	[∡→] →GRAD	
→HEX	INTS [▼] →HEX	[4] (1) {1, 2} → {1} Converts a closed number to a hexad. *short integer*.
→HR	CATALOG FCNS →HR	(1) {5} → {2} Operates on *times* like →REAL below. Maintained for backward compatibility only.

[62] Cf. MOD.

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
→H.MS	h.ms for closed x	[2] (1) {1, 2, 5} → {5} Converts a decimal x to a sexagesimal *time* – cf. p. 166. Does not change a sexagesimal *time*.
→INT	# ***base*** for closed x	[2+] (1) {1, 2, 10} → {10} Converts IP(x) to a *short integer* of the base specified. Conversion to decimal may be abbreviated by #D, to hexadecimal by #H. Note #I returns a *long integer* instead (cf. p. 48).
→MIL	→MIL	[3] (1) {1, 2, 4} → {4} Converts *angles* as described on pp. 161f.
→MULπ	→MULπ	
→OCT	INTS ▼ →OCT	[4] (1) {1, 2} → {1} Converts a closed number to an octal *short integer*.
→POL	→P	[2] {2}; {1} → {2}  Assumes **X** and **Y** containing *2D Cartesian* coordinates of a point or components of a vector (***x*** , ***y***). Converts them to the respective polar coordinates or components (***r***, ***ϑ***). Inverted by →REC. For switching the display format of *complex numbers*, see POLAR on p. 128.
→RAD	→RAD	[3] (1) {1, 2, 4} → {4} Converts *angles* as described on pp. 161f.
→REAL	.d for closed x	[2] (1) {1, 2, 4, 5, 6, 10} → {2} Converts x to a *real number*. Any object displayed sexagesimal (e.g., a *time*) will be converted in a decimal number and shown in the format set. For *dates*, the date format chosen is taken into account. Fraction display will return to decimal numbers (cf. FRACT and PROPFR). To return the *real* part of a *complex* object, take RE (cf. p. 74). To cut a *complex* object in its two parts, use CC or CX→RE (cf. p. 33).

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
→REC	R←	[2] {2}; {1, 4} → {2} Assumes **X** and **Y** containing *2D* polar coordinates of a point or components of a vector (***r***, ***ϑ***). Converts them to the respective *Cartesian* coordinates or components (***x***, ***y***). Inverted by →POL. For switching the display format of *complex* objects, see POLAR on p. 128.
↑Lim ↓Lim	ADV ∫fdx ↑Lim etc.	[4] Reserved *real variables* for the upper and lower limit of integration (see *OMS 4*).
⥮	STK ⥮____ r_1, r_2, r_3, r_4	[3++++] Shuffles the contents of the four bottom *stack registers* **X**, **Y**, **Z**, and **T** at execution time. **Examples:** ⥮xxyz works like ENTER↑ (but enables *ASL*!), ⥮yxzt works like x⇆y, ⥮yztx works like R↓ in a *4-level stack*, ⥮txyz works like R↑ in a *4-level stack*, but also ⥮yytt or ⥮zzzx are possible. **ATTENTION:** This is a very powerful command although it does not look it. Note it will affect the bottom 4 *stack registers* only; there is no connection to **A** … **D**, regardless of *stack* size. Playing with ⥮, you may lose *stack* contents and make a mess of the *stack* easily.
|M|	MATX |M|	[3] (1) {8} → {2}; {9} → {3} Requires a square matrix in **X** and returns its determinant. The original matrix is saved in **L**.
|x|	|x| or PART |x| or CPX |x|	[2] (1) {1, 2, 4, 10} Returns the absolute (unsigned) value of ***x***. (1) {3} → {2} Returns the *magnitude* $\sqrt{\mathrm{Re}(x)^2+\mathrm{Im}(x)^2}$ in **X**. (1) {8*}; {9*} → {8} Returns a *real* matrix with the absolute values or *magnitudes* of all matrix elements. Cf. ENORM.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
||	[||]	[2] (2) {2, 3}; {1} → {2} Returns $\left({}^{1}/_{x} + {}^{1}/_{y}\right)^{-1}$; useful in electrical engineering especially. Returns 0. for $x\,y = 0$.
%	[FIN] [%]	[3] (1) {2}; {1} → {2} Returns ${}^{xy}/_{100}$, leaving y unchanged.
%ile	[STAT] [▲] [%ile]	[4] (−2) { } → {2} Pushes the *x%-percentile* of the accumulated y- and x-data on the *stack*. Note this may only make sense for data sets containing ≥ 50 samples.
%MRR	[FIN] [%MRR]	[3] (3) {2}; {1} → {2} Returns the *mean rate of return*[63] in % per period, i.e. $100 \cdot \left(\sqrt[z]{{}^{x}/_{y}} - 1\right)$ with x = FV = future value after z periods, and y = PV = present value. For z = 1, Δ% returns the rate of return easier.
%T	[FIN] [%T]	[3] (1) {2}; {1} → {2} Returns $100\ {}^{x}/_{y}$, interpreted as % of total. Leaves y unchanged.
%Σ	[FIN] [%Σ]	[3] (1) {2}; {1} → {2} Returns $100\ x\ /\ \sum x$.
%+MG	[FIN] [%+MG]	[3] (2) {2}; {1} → {2} Calculates a sales price by adding a margin of x % to the cost y. Formula: $P_{sale} = y\ /\left(1 - \frac{x}{100}\right)$ You may use %+MG for calculating net amounts as well; just enter a negative percentage in x.
√x	[√x]	[1] (1) {1, 2, 3, 8*, 9*, 10}; ({1} → {2}, {1, 2} → {3}) Returns the square root of x.
∞		Constant (see pp. 113ff).

[63] *TNG*: *MRR* = mittlere Rendite (über mehrere Perioden).

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
∫	ADV ∫fdx ∫ ***label*** (called in *RUM*)	[4+] {2} Integrates the expression specified in the routine ***label***. See common remarks for more. **ATTENTION:** ∫ fills all *stack registers* with x before calling the routine specified.
	ADV ∫fdx ∫ ***var*** (called in *PEM*)	[4+] {2} Integrates the expression given in the routine specified by PGMINT over the *integration variable* ***var***. Listed in programs as ∫fd followed by ***var***. See common remarks for more. **ATTENTION:** ∫ fills all *stack registers* with x before calling the routine specified.
	EQN ∫f ∫	[4] Integrates the current equation.
		Common remarks: Lower and upper integration limits must be supplied by the corresponding *variables* ↓Lim and ↑Lim, accuracy by ACC. ∫ returns the (approximate) integral in **X** and an upper limit of its uncertainty in **Y**. **Z** is cleared. Higher *stack registers* are not affected. See *OMS 4* for examples and *App. J* for more information. ∫ may be aborted by EXIT.
∫f	EQN ∫f	*Submenus*. See pp. 140f.
∫fdx	ADV ∫fdx	
∡	∠ or CPX ∡ or PART ∡	[2] (1) {2} → {4} Returns 180° (or equivalent) for $x < 0$ and 0 else.
		(1) {3} → {4} Returns the *phase* or argument (cf. |x|) $$arg(x) = arctan\left(\frac{Im(x)}{Re(x)}\right)$$
		(1) {9*} → {8} Returns a matrix with the *phases* of all elements of the input matrix. Cf. |x|.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
∡→	∡→	*Menu* of angular conversions. See p. 149.
🖶ADV	🖶ADV	(0) Prints the current contents of the print buffer and a linefeed. **ATTENTION:** The printer will print only when a linefeed is sent to it.
🖶CHAR	🖶CHAR ***n***	(0) Sends a single *character* (with the code specified) to the printer. *Character codes* ***n*** > 127 can only be specified indirectly. 🖶MODE setting will be honored. Cf. 🖶ADV.
🖶DLAY	🖶DLAY ***n***	(0) Sets a delay of ***n*** ticks (see TICKS) to be used with each linefeed on the printer.
🖶MODE	🖶MODE ***n***	(0) Sets print mode. Legal print modes are: 0: Use the printer font and *character* set wherever possible (default). All *characters* feature the same width (5 columns + 2 columns spacing). 1: Use the variable pitch display font, resulting in some jitter on the printout but packing more *characters* in a row. 2: Use the small display font, which allows for packing even more info in a row. 3: Send the output to the serial line. Works for plain ASCII only – no *characters* will be translated. Line setup is the same as for serial communication: 9600 baud, 8 bits, no parity.
🖶PROG	🖶PROG	(0) Prints the listing of the *current program* (see *OMS 3*), one row per step.
🖶r	🖶r ***s***	(0) Prints the content of the *RoV* specified, right adjusted, without labeling the output. If you want a heading label, compose the string in **X** first or use 🖶REGS. Cf. 🖶ADV.
🖶REGS	🖶REGS	(0) Interprets x in the form `sss.nn` . Prints the contents of ***nn*** *registers* starting with number ***sss*** . Each *register* takes one row starting with its label.

DRAFT

Item	Keystrokes	Remarks (cf. pp. 14ff for general information)
		ATTENTION for ***nn*** = 0 : • For $sss \in [0;99]$, printing will stop at R99. • For $sss \in [100;111]$, printing stops at K. For $sss \geq 112$, printing stops at the highest allocated local *register*.
🖶STK	🖶STK	(0) Prints the entire *stack* contents. Each of the 4 or 8 *registers* prints in a separate row starting with its label.
🖶TAB	🖶TAB ***n***	(0) Positions the print head to print column ***n*** (0 to 165, where ***n*** > 127 can only be specified indirectly). Useful in formatting (in 🖶MODE 1 or 2 in particular). Allows also for printer plots. If ***n*** is less than current print head position, a linefeed will be entered to reach the new position. Cf. 🖶ADV.
🖶USER	🖶USER	(0) Prints all *variables* and *global* program *labels* in alphabetic order. The *variables* are printed first; if you are not interested in the program *labels*, press R/S to stop the listing.
🖶WIDTH	🖶WIDTH	(−1) Returns the number of print columns that ***x*** would take in the print mode set. Cf. 🖶MODE. Second use: in 🖶MODE 1 or 2, 🖶WIDTH returns the width of ***x*** in px (including the last column being always blank) in the specified font.
🖶x	🖶x	(0) Prints ***x*** without a heading label. Cf. 🖶ADV.
🖶Σ	🖶Σ	(0) Prints the summation *registers*. Each *register* prints in one row starting with its label.
🖶#	🖶# ***n***	(0) Sends a single *byte*, without translation, to the printer (e.g., a control code). ***n*** > 127 can only be specified indirectly. 🖶MODE setting will not be honored. Cf. 🖶ADV.
#B	BIT #B	[4] (1) {10} Counts the bits set in ***x*** (like on *HP-16C*).
#DEC		Reserved *integer variable* for the parameter specified in last call of ALL, FIX, SCI, ENG, or DSP).

DRAFT

Constants

Press CONST to access a *catalog* of 79 physical, astronomical, and mathematical constants sorted alphabetically. *Names* of astronomical (incl. surveying) and mathematical constants are printed on colored background in the table starting overleaf. Values of fundamental physical constants incl. their relative *SDs* below are printed on white if they are exactly defined – the darker grey the background, the less precisely the particular value is known.[64] Constants defining standard conditions are printed on orange.

In this chapter, we use commas as radix marks for better visibility and multiplication dots for space reasons. Formulas are printed where applicable.

[64] For most physical constants in CONST, their precise numeric values and their relative *SDs* are from *CODATA 2022*, copied in April 2025. These are the best values known in the scientific community today, agreed on by the national standards institutes worldwide (e.g., by *NIST* and *PTB*). Note that the six fundamental constants (printed in **bold** in the table) all feature less than 11 significant digits.

Relative SDs are included in this table here though not contained in CONST. They are important for computing the precision of results you obtain using these constants given, through the process of *error propagation* (see *App. I*) going back to *Carl F. Gauss* (1777 - 1855). This is essential for trustworthy results – not only in science (remember each and every scientific result shall include an indication of its uncertainty). Please consult suitable reference (e.g., *http://physics.nist.gov/cgi-bin/cuu/Info/Constants/definitions.html*, giving a nice introduction, and also *https://physics.nist.gov/cuu/Uncertainty/index.html*). There is simply no way yardstick measurements can yield results accurate to four decimals.

Apropos, the terms *resolution*, *precision*, and *accuracy* are confused frequently in measuring (and hence in advertising all the more, sometimes on purpose). In a nutshell, *resolution* is the least significant digit a measuring instrument indicates. Using this instrument to measure the same object or property under identical conditions multiple times, you get an idea of its *repeatability* (or *precision*); this will include your interaction with said instrument and can be no better than its *resolution* but may be significantly worse – a factor of ten or more may be observed in real life. *Accuracy* of a measuring instrument can never be any better than its *repeatability*, however.

Since we can't know anything about any real-life object or process any better than we can measure it, these considerations are fundamental. We recommend watching them to distinguish truth from fake – in your very own interest.

DRAFT

Name	Numeric value and *rel. SD*	Remarks
a (0)[65]	365,242 5 d (*per definition*)	*Gregorian* year
a_0	$5{,}291\,772\,105\,44 \cdot 10^{-11}$ m *(1,6 · 10⁻¹⁰)*	*Bohr* radius $a_0 = \alpha / 4\pi R_\infty$
a_{Moon}	$3{,}844 \cdot 10^{8}$ m *(1 · 10⁻³)*	Semi-major axis of the Moon's orbit around the Earth ≈ 1,3 *light seconds*.
$a_\oplus$	$1{,}495\,979 \cdot 10^{11}$ m *(1 · 10⁻⁶)*	Semi-major axis of the Earth's orbit around the Sun. Within the uncertainty stated here, it equals 1 *astronomic unit* ≈ 499 *light seconds* ≈ 8 *light minutes*.
c	**2,997 924 58 · 10⁸ m/s**	Speed of light in vacuum $\approx 300\,000\ \mathrm{km/s} = 300\ \mathrm{km/ms} = 300\ \mathrm{m/\mu s} = 30\ \mathrm{cm/ns} = 0{,}3\ \mathrm{mm/ps} = 300\ \mathrm{nm/as}$.[66]
c_1 (5)	$3{,}741\,771\,85\ldots \cdot 10^{-16}$ W m²	First radiation constant $c_1 = 2\pi h c^2$
c_2	0,014 387 768 77… m·K	Second radiation constant $c_2 = hc/k$
e	**1,602 176 634 · 10⁻¹⁹ A s**	Elementary charge $e = \frac{2}{K_J R_K} = \Phi_0 G_0$
e	2,718 281 828 459 045 2…	*Euler's* e.
F	96 485,332 12… A s/mol	*Faraday* constant $F = e\,N_A$
F_α (10)	2,502 907 875 095 892 8...	*Feigenbaum's* α and δ
F_δ	4,669 201 609 102 990 6...	
G	$6{,}674\,30 \cdot 10^{-11}$ m³/kg s² *(2,2 · 10⁻⁵)*	*Newtonian* constant of gravitation; also known as γ from other authors. See $GM_\oplus$ for a more precise value.

[65] The numbers in parentheses in this column are to support determination of parameters for CNST – cf. the *IOI*.

[66] If you need some 0,2 … 10 ns fix delays for electronic signals, you can achieve these using coaxial cables easily. At the long end you will need some space, though.

Name	Numeric value and *rel. SD*	Remarks
G_0	7,748 091 729…· 10^{-5} / Ω	Conductance quantum $G_0 = {2e^2}/{h} = {2}/{R_K} = eK_j$
G_C	0,915 965 594 177 219 0…	*Catalan's* constant
g_e (15)	−2,002 319 304 360 92 *(1,8 · 10^{-13})*	*Landé's* electron g-factor
$GM_\oplus$	3,986 004 418 · 10^{14} ${m^3}/{s^2}$ *(per definition)*	*Geocentric gravitational constant* incl. the mass of the atmosphere (according to the Earth model *WGS84*) [67]
$g_\oplus$	9,806 65 ${m}/{s^2}$ *(per definition)*	Standard Earth acceleration (actual values vary from 9,780 4 ${m}/{s^2}$ at the equator to 9,832 2 ${m}/{s^2}$ at the poles)
h	**6,626 070 15 · 10^{-34} J s**	*Planck* constant
$\hbar$ (19)	1,054 571 817…· 10^{-34} J s	Reduced *Planck* constant $\hbar = {h}/{2\pi}$
i		$= \sqrt{-1}$
k (20)	**1,380 649 · 10^{-23} ${J}/{K}$**	*Boltzmann* constant $k = {R}/{N_A}$
K_J	4,835 978 48…· 10^{14} ${Hz}/{V}$	*Josephson* constant $K_J = {2e}/{h}$
l_{PL}	1,616 255 · 10^{-35} m *(1,1 · 10^{-5})*	*Planck* length $l_{PL} = t_{PL}c$
m_e	9,109 383 713 9 · 10^{-31} kg *(3,1 · 10^{-10})*	Electron rest mass ≙ 511,00 keV [68]
M_{Moon}	7,349 · 10^{22} kg *(5 · 10^{-4})*	Mass of the Moon
m_n (25)	1,674 927 500 56 · 10^{-27} kg *(5,1· 10^{-10})*	Neutron rest mass ≙ 939,57 MeV

[67] See https://earth-info.nga.mil/php/download.php?file=coord-wgs84 (207 pages).

[68] For comparison: the mass of the (electron) neutrino is $0{,}02\ \text{eV} \leq m_\nu \leq 0{,}8\ \text{eV}$.

Name	Numeric value and *rel. SD*	Remarks
m_n/m_p	1,001 378 419 46 *(4,0 · 10⁻¹⁰)*	Neutron to proton rest mass ratio
m_p	1,672 621 925 95 · 10⁻²⁷ kg *(3,1 · 10⁻¹⁰)*	Proton rest mass ≙ 938,27 MeV
m_{PL}	2,176 434 · 10⁻⁸ kg *(1,1 · 10⁻⁵)*	*Planck* mass $m_{PL} = \sqrt{\hbar c / G} \approx 22\ \mu g$
m_p/m_e	1 836,152 673 426 *(1,7 · 10⁻¹¹)*	Proton to electron rest mass ratio
m_u (30)	1,660 539 068 92 · 10⁻²⁷ kg *(3,1 · 10⁻¹⁰)*	Atomic mass constant $\approx 10^{-3}\ kg/N_A$
$m_u c^2$	1,492 418 087 68 · 10⁻¹⁰ J *(3,1 · 10⁻¹⁰)*	Energy equivalent of the atomic mass constant ≈ 931,49 MeV
m_μ	1,883 531 627 · 10⁻²⁸ kg *(2,2 · 10⁻⁸)*	Muon (myon) rest mass ≙ 105,66 MeV
$M_\odot$	1,989 1 · 10³⁰ kg *(5 · 10⁻⁵)*	Mass of the Sun (or the Solar system in total)
$M_\oplus$	5,973 6 · 10²⁴ kg *(5 · 10⁻⁵)*	Mass of the Earth. Cf. $GM_\oplus$ for a more precise value.
N_A (35)	**6,022 140 76 · 10²³ / mol**	*Avogadro's* number
NaN	*Not a Number*	**NaN** covers poles as well as regions where a function result is not defined at all (infinities, on the other hand, are considered numeric by your *WP43* – see below). Unless SPCRES is set, non-numeric results lead to an error message thrown – **NaN** allows that functions written by you can return it instead. See also p. 194.
p_0	101 325 Pa *(per definition)*	Standard atmospheric pressure
R	8,314 462 618… J/mol K	Molar gas constant $R = k\,N_A$

Name	Numeric value and *rel. SD*	Remarks
r_e	2,817 940 320 5 · 10^{-15} m *(4,7 · 10^{-10})*	Classical electron radius $r_e = \alpha^2 \, a_0$
R_K (40)	25 812,807 45… Ω	*Von Klitzing* constant $R_K = {}^{h}/_{e^2}$
R_{Moon}	1,737 530 · 10^6 m *(5 · 10^{-7})*	Mean radius of the Moon
R_∞	10 973 731,568 157 / m *(1,1 · 10^{-12})*	*Rydberg* constant $R_\infty = {}^{\alpha^2 \, m_e \, c}/_{2h}$
$R_\odot$	6,96 · 10^8 m *(5 · 10^{-3})*	Mean radius of the Sun
$R_\oplus$	6,371 010 · 10^6 m *(5 · 10^{-7})*	Mean radius of the Earth
Sa (45)	6,378 137 0 · 10^6 m *(per definition)*	Semi-major axis[69] (cf. the link in f. 67 on p. 115 for more)
Sb	6,356 752 314 2 · 10^6 m	Semi-minor axis[69] $S_b = S_a \left(1 - {}^{1}/_{S_{f^{-1}}}\right)$
Se^2	6,694 379 990 141 · 10^{-3}	1st eccentricity squared[69] $S_e^2 = 1 - \frac{S_a^2}{S_b^2}$
Se'^2	6,739 496 742 276 · 10^{-3}	2nd eccentricity squared[69] $S_{e'}^2 = \frac{S_a^2 - S_b^2}{S_b^2}$
Sf^{-1}	298,257 223 563 *(per def.)*	Flattening factor [69]
T_0 (50)	273,15 K *(per definition)*	= 0°C, standard temperature
T_{PL}	1,416 784 · 10^{32} K *(1,1 · 10^{-5})*	*Planck* temperature $T_P = \frac{c^2}{k}\sqrt{\frac{\hbar c}{G}} = \frac{M_P c^2}{k} = \frac{E_P}{k}$
t_{PL}	5,391 247 · 10^{-44} s *(1,1 · 10^{-5})*	*Planck* time $t_{PL} = {}^{l_{PL}}/_{c}$

([69] ... of the Earth according to *WGS84*. See the linked publication (cf. f. 67) also for the uncertainties of **Sb**, **Se^2**, and **Se'^2**.

Name	Numeric value and *rel. SD*	Remarks
V_m	0,022 413 969 5… $^{m^3}/_{mol}$	Molar volume of an ideal gas at standard conditions $V_m = \frac{R\,T_0}{p_0} \approx 22{,}4\ {}^{l}/_{mol}$
Z_0	376,730 313 412 Ω *(1,6 · 10⁻¹⁰)*	Characteristic impedance of vacuum
α (55)	7,297 352 564 3 · 10⁻³ *(1,6 · 10⁻¹⁰)*	Fine-structure constant $\alpha = \frac{e^2}{2\varepsilon_0\, h\, c} \approx \frac{1}{137}$
γ	6,674 30 · 10⁻¹¹ $^{m^3}/_{kg\ s^2}$ *(2,2 · 10⁻⁵)*	*Newtonian* constant of gravitation; also known as **G** from other authors. Cf. $GM_\oplus$ for a more precise value.
γ_{EM}	0,577 215 664 901 532 9…	*Euler's* (a.k.a. *Euler-Mascheroni*) constant
γ_p	2,675 221 870 8 · 10⁸ $^{Hz}/_{T}$ *(4,3 · 10⁻¹⁰)*	Proton gyromagnetic ratio $\gamma_p = 4\pi\,\frac{\mu_p}{h}$
$\Delta\nu_{Cs}$	**9 192 631 770 Hz**	Hyperfine transition frequency of ^{133}Cs
ε_0 (60)	8,854 187 818 8 · 10⁻¹² $\frac{As}{Vm}$ *(1,6 · 10⁻¹⁰)*	Vacuum electric permittivity $\varepsilon_0 = {}^{1}/_{\mu_0\, c^2}$ (note the so-called *Coulomb's* constant is just $^{1}/_{4\pi\varepsilon_0}$)
λ_C	2,426 310 235 38 · 10⁻¹² m *(3,1 · 10⁻¹⁰)*	*Compton* wavelengths of the electron $\lambda_C = {}^{h}/_{m_e c}$, neutron $\lambda_{Cn} = {}^{h}/_{m_n c}$, and proton $\lambda_{Cp} = {}^{h}/_{m_p c}$, respectively
λ_{Cn}	1,319 590 903 82 · 10⁻¹⁵ m *(5,1 · 10⁻¹⁰)*	
λ_{Cp}	1,321 409 853 60 · 10⁻¹⁵ m *(3,1 · 10⁻¹⁰)*	
μ_0	1,256 637 061 27 · 10⁻⁶ $\frac{Vs}{Am}$ *(1,6 · 10⁻¹⁰)*	Vacuum magnetic permeability

DRAFT

Name	Numeric value and *rel. SD*	Remarks
μB (65)	9,274 010 065 7 · 10⁻²⁴ J/T *(3,1 · 10⁻¹⁰)*	*Bohr's* magneton $\mu_B = e\hbar/2m_e$
μe	−9,284 764 691 7 · 10⁻²⁴ J/T *(3,1 · 10⁻¹⁰)*	Electron magnetic moment
μe/μB	−1,001 159 652 180 46 *(1,8 · 10⁻¹³)*	Ratio of μ_e to *Bohr's* magneton
μn	−9,662 365 3 · 10⁻²⁷ J/T *(2,4 · 10⁻⁷)*	Neutron magnetic moment
μp	1,410 606 795 45 · 10⁻²⁶ J/T *(4,3 · 10⁻¹⁰)*	Proton magnetic moment
μu (70)	5,050 783 739 3 · 10⁻²⁷ J/T *(3,1 · 10⁻¹⁰)*	Nuclear magneton $\mu_u = e\hbar/2m_p$
μμ	−4,490 448 30 · 10⁻²⁶ J/T *(2,2 · 10⁻⁸)*	Muon (myon) magnetic moment
σB	5,670 374 41…· 10⁻⁸ $\mathrm{W/m^2K^4}$	*Stefan-Boltzmann* constant $\sigma_B = \frac{2\,\pi^5\,k^4}{15\,h^3\,c^2}$
Φ	1,618 033 988 749 894 8…	Golden ratio $\Phi = (1+\sqrt{5})/2$
Φ0	2,067 833 848…· 10⁻¹⁵ V s	Magnetic flux quantum $\Phi_0 = h/2e$
ω (75)	7,292 115 · 10⁻⁵ rad/s *(per definition)*	*Nominal mean angular velocity of the Earth* according to *WGS84* (cf. p. 115 for f. 67).
-∞	−∞	Note both are counted as numeric values in your *WP43.* Unless SPCRES is set, an error will be thrown for infinite results. See also p. 194.
∞	∞	

DRAFT

Conversion factors follow in this table:

	0.	
	0,352 777 …	Point → mm
(80)	25,4	Inch → mm
	0,304 8	Foot → m
	0,304 800 6…	Survey foot → m
	0,9144	Yard → m
	1,828 8	Fathom → m
(85)	1,609 344	Mile → km
	1,852	Nautical mile → km
	$1,495\ 97\ldots \cdot 10^{11}$	Astronomical units → m
	$9,460\ 73\ldots \cdot 10^{15}$	Light year → m
	$3,085\ 67\ldots \cdot 10^{16}$	Parsec → m
(90)	500,0	Li → m
	0,03	Yin → m
	30,0	Cun → m
	0,3	Zhang → m
	300,0	Fen → m
(95)	1 609,344	Mile → m
	1 852,0	Nautical mile → m
	0,025 4	Inch → m
	$3,527\ 77\ldots \cdot 10^{-4}$	Point → m
	0,404 685 …	Acre → ha
(100)	0,404 687 …	Acre (US) → ha
	0,001 5	Mu → m^2
	40 468,56…	Acre → m^2
	40 468,72…	Acre (US) → m^2
	28,413 06…	Fluid ounces (UK) → ml
(105)	29,573 5…	Fluid ounces (US) → ml
	3,785 41…	Gallons (US) → l
	4,546 09	Gallons (UK) → l
	1,136 522 …	Quart (UK) → l
	0,158 987 …	Barrel → m^3
(110)	$2,841\ 30\ldots \cdot 10^{-5}$	Fluid ounces (UK) → m^3
	$2,957\ 35\ldots \cdot 10^{-5}$	Fluid ounces (US) → m^3
	$3,785\ 41\ldots \cdot 10^{-3}$	Gallons (US) → m^3
	$4,546\ 09 \cdot 10^{-3}$	Gallons (UK) → m^3

DRAFT

	1,136 52 …·10^{-3}	Quart (UK) → m^3
(115)	0,2	Carat → g
	28,349 5…	Ounces → g
	31,103 4…	Troy ounces → g
	0,453 592 …	lbs → kg
	6,350 29…	Stone → kg
(120)	45,359 2…	Short cwt → kg
	50,802 3…	cwt → kg
	907,184 …	Short ton → kg
	1 016,046 …	Ton → kg
	0,000 2	Carat → kg
(125)	28,349 5…	Ounces → kg
	31,103 4…	Troy ounces → kg
	4,186 8	Calories → J
	1 055,05…	Btu → J
	3 600,0	Wh → J
(130)	4,448 22 …	lbf → N
	133,322 …	torr → Pa
	133,322 …	mmHg → Pa
	3 386,38…	Inches Hg → Pa
	6 894,75…	psi → Pa
(135)	100 000,0	bar → Pa
	101 325,0	atm → Pa
	735,498 …	hpm → W
	745,699 …	hpUK → W
	746,0	hpe → W
(140)	31 556 952,0	Year → s
	1,355 817 …	lbfft → Nm
_108	−108,0	Auxiliary constants start here
_4	−4,0	
_1	−1,0	

The table continues with more auxiliary constants, printed in three consecutive columns for each page following:

DRAFT

(145)	_1oneE	−0,367 8…	(180)			(215)	
	_7on20	−0,35		4		2100	
	_1on4	−0,25		3piOn2	4,712 38…	2520	
	1e_49	10^{-49}		5		2916	
	1e_37			6		3600	
(150)	1e_24		(185)	2pi	6,283 18…	(220) 6000	
	1e_16			7		6400	
	1e_6			8		9000	
	1e_4			9		9999	
	9on160	0,056 25		10		10000	
(155)	1on10	0,1	(190)	12		(225) 25200	
	1on8	0,125		16		42000	
	5on24	0,208 33…		20		73766	
	111on500	0,222		24		86400	
	1on4	0,25		25		1e6	
(160)	33on125	0,264	(195)	27		(230) 1e8	
	1on3	0,333 …		29		1e9	
	1on2	0,5		30		2p32 (=2^{32})	$4{,}29\ldots\cdot10^{9}$
	2on3	0,666 …		gammaR		1e32	
	egamma	0,577 21…		32		1e6143	
(165)	ln2	0,693 14…	(200)	36		(235) 1e_10000	
	root2on2	0,707 10…		47		995on1000	0,995
	piOn4	0,785 39…		54		1e_32	
	root3on2	0,866 02…		180onPi	57,295 7…	TspukToMl[70]	5,919 38…
	9on10	0,9		60		TbspukToMl	17,758 1…
(170)	ln2piOn2		(205)	200onPi	63,025 3…	CupukToMl	284,130 …
	1			125		PintukToMl	568,261 …
	3on2	1,5		150		QuartukToMl	1 136,52…
	piOn2	1,570 79…		205		GalukToMl	4 928,92…
	9on5	1,8		360		TspusToMl	4,928 92…
(175)	19on10	1,9	(210)	400		TbspusToMl	14,786 7…
	2			600		CupusToMl	236,588 …
	ln10	2,302 58…		1000		PintusToMl	473,176 …
	3piOn4	2,356 19…		3200onPi	1018,59…	QuartusToMl	946,352 …
	3			1260		QuartusToL	0,946 35…
(180)	pi	3,141 59…	(215)	1680		GalusToMl	3 785,41…

[70] The following kitchen conversions were added later than the other conversions.

(250)		(285)		(320)		
TspmToMl	5,0	gammaC19			ln2	0,693 14…
TbspmToMl	15,0	gammaC20			1	
CupmToMl	250,0	gammaC21			piOn2	1,570 79…
TbspauToMl	20,0	gammaC22			2	
PintdryToL	0,550 61…	(290) gammaC23		(325)	ln10	2,302 58…
QuartdryToL	1,101 22…	gammaC24			3	
GaldryToL	4,404 88…	gammaC25			pi	3,141 59…
PeckusToL	8,809 76…	gammaC26			4	
BushelusToL	35,239 0…	gammaC27			5	
Inch3ToMl	16,387 0…	(295) gammaC28		(330)	2pi	6,283 18…
Ft3ToL	28,316 8…	gammaC29			7	
FlozukToInch3	1,733 87…	pi_51	3,141 59…		9	
FlozusToInch3	1,804 68…	200			10	
Ft3ToGaluk	6,228 83…	180			12	
Ft3ToGalUS	7,480 51…	(300) 3200		(335)	14	
QuartusToM3	0,000 94…	piOn2_51	1,570 79…		24	
gammaC00	Lanczos's	100			28	
gammaC01	coefficients	90			31	
gammaC02		1600			38	
(270) gammaC03		(305) piOn4_51	0,785 39…	(340)	47	
gammaC04		50			60	
gammaC05		45			100	
gammaC06		800			153	
gammaC07		2pi [71]	6,283 18…		275	
(275) gammaC08		(310) _4712	−4712,0	(345)	367	
gammaC09		_1			400	
gammaC10		0			1000	
gammaC11		1e_32			1401	
gammaC12		1e_6			1461	
(280) gammaC13		(315) 1e_4		(350)	3600	
gammaC14		1on100	0,01		4716	
gammaC15		1on10	0,1		4800	
gammaC16		1on3	0,333 33…		4900	
gammaC17		1on2	0,5		5001	
(285) gammaC18		(320) 2on3	0,666 66…	(355)	10000	

[71] At CNST(309), the first 1100 decimals of 2π are stored (and many more can be read there).

(355)		(360)		(365)	
20000		146097		maxDate	$3{,}1\ldots\cdot10^{19}$
32075		274277		maxTime	$3{,}6\cdot10^{19}$
43200		1e6			
65535		1729777			
(360)	86400	(365)	2p32	$4{,}29\ldots\cdot10^{9}$	

In total, 368 constants are addressable by CNST. They are stored in *QSPI* memory (see *App. A*). Each of the named constants will appear in programs with a heading # (e.g., `# c`) as shown in *OMS 3*.

A few constants more follow for your convenience:

Numeric value	Remarks
$11{,}2\cdot10^{3}\ {}^{\mathrm{m}}/_{\mathrm{s}}$	Escape speed from Earth
$343\ {}^{\mathrm{m}}/_{\mathrm{s}}$	Speed of sound in … dry air at standard lab conditions (101,325 kPa, 20°C)[72]
$1{,}2041\ \mathrm{kg/m^3}$	Density of
$28{,}965\cdot10^{-3}\ \mathrm{kg/mol}$	Molar mass of
$1370\ {}^{\mathrm{W}}/_{\mathrm{m}^2}$	Solar constant, i.e. average incident power at the mean distance of Earth to Sun outside the atmosphere
$1{,}62\ {}^{\mathrm{m}}/_{\mathrm{s}^2}$	Gravity acceleration on the Moon
$13{,}8\cdot10^{9}\ \mathrm{a}$	Age of the universe ($=4{,}35\cdot10^{17}\ \mathrm{s}$)
$46{,}5\cdot10^{9}\ l.y.$	Radius of the observable universe[73] ($\approx 4{,}40\cdot10^{26}\ \mathrm{m}$)

[72] See https://en.wikipedia.org/wiki/Density_of_air#Dry_air

[73] For more information, please see https://en.wikipedia.org/wiki/Observable_universe (or https://de.wi.../Beobachtbares_Universum or https://fr.wi.../Univers_observable or https://es.wikipedia.org/wiki/Universo_observable etc. – the latter site includes a picture https://es.wikipedia.org/wiki/Universo_observable#/media/Archivo:Universoobservable.PNG explaining the difference between the 46.5×10⁹ *light years* as printed above and the naïve assumption of 13.8×10⁹ *light years* for the radius; this picture was not found on other sites checked).
Thus, the circumference of the observable universe is $292\cdot10^{9}\ l.y.\approx 3\cdot10^{27}\ \mathrm{m}$. Note the uncertainty caused by the finite precision of your *WP43* means just a deviation of $3\cdot10^{-7}\ \mathrm{m}=0.3\ \mu\mathrm{m}$ in this circumference of 292 billion *light years*! Think about its

DRAFT

These are not stored in CONST. Of some further constants, only 1 or 2 digits are known – they may help you in quick estimates nevertheless:

Radius of an atomic nucleus[74]	$\sim 10^{-15}$ m
Radius of an atom[75]	$\sim 10^{-10}$ m
Radius of our home galaxy	$\sim 10^{5}\ l.y. \approx 10^{21}$ m
Hubble 'constant' [76]	$\sim 70 \frac{\text{km/s}}{\text{Mpc}} \approx {}^{1}/_{4{,}4 \cdot 10^{17}}$ s
Baryonic mass in observable universe[77]	$\sim 10^{53}$ kg
Number of stars in our home galaxy[78]	$\sim 2 \cdot 10^{11}$
Number of neuron connections in human brain	$\sim 10^{14}$
Number of stars in observable universe	$\sim 10^{23}$
Number of atoms in a human of 75 kg	$\sim 7.2 \cdot 10^{27}$
Number of legal board positions in chess	$\sim 10^{43}$

relevance.

[74] E.g., the (charge) radius of an hydrogen nucleus (i.e. a proton) is 0,840 9(4) fm, of helium 1,678 24(83) fm (data of 2025-06). Just in case you are looking for something really small.

Actually, helium is used for *UHV* leak testing – though helium atoms since bare helium nuclei are more difficult to generate. Helium is the smallest atom nature provides (its radius is 28 pm), hydrogen is slightly bigger (32 pm), then comes neon with 58 pm.

[75] So the nucleus takes far less than a billionth of the volume of an atom. Electrons are even smaller. Thus, an atom is almost completely empty space. On the other hand, our world as we know and watch it every day is built of atoms. You can even touch many of these real-world objects. Think about it.

These facts also give some hand-waving arguments why cancer (tumor) therapy using heavy ion beams works at all (and often significantly better than using X-rays).

[76] Two significantly different values are known: 73 and 67,4. Discussions are ongoing in 2025 still. – Note this 'constant' does change over very long times actually.

[77] This is the mass of all common matter (i.e. atoms, ions, molecules, etc.). Physicists think this can't be everything (for good reasons) and talk about 'dark matter' – but the latter is not detected yet although a dark galaxy was found recently (2025-04).

[78] Gedankenexperiment: Assume our home galaxy fills a huge disk-like cylinder 1000 $l.y.$ high, what will be the average distance between its stars? So what do you expect to happen when two such galaxies collide? Calculate yourself, then look up f. 106.

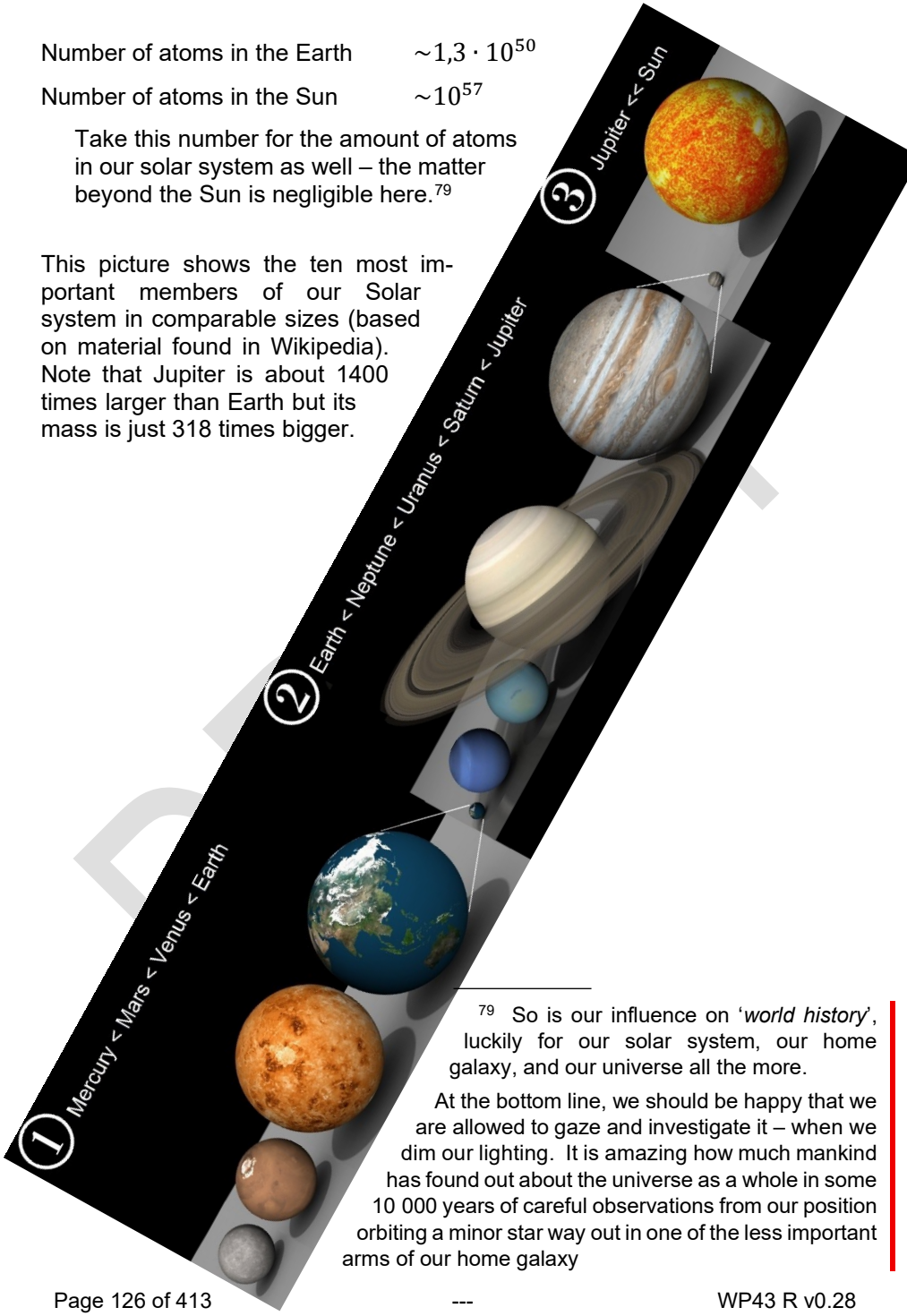
DRAFT

Number of atoms in the Earth $\sim 1{,}3 \cdot 10^{50}$

Number of atoms in the Sun $\sim 10^{57}$

Take this number for the amount of atoms in our solar system as well – the matter beyond the Sun is negligible here.[79]

This picture shows the ten most important members of our Solar system in comparable sizes (based on material found in Wikipedia). Note that Jupiter is about 1400 times larger than Earth but its mass is just 318 times bigger.



[79] So is our influence on '*world history*', luckily for our solar system, our home galaxy, and our universe all the more.

At the bottom line, we should be happy that we are allowed to gaze and investigate it – when we dim our lighting. It is amazing how much mankind has found out about the universe as a whole in some 10 000 years of careful observations from our position orbiting a minor star way out in one of the less important arms of our home galaxy

Number of hydrogen atoms in observable universe [80]	$\sim 10^{80}$
Number of legal board positions in the Japanese game Go	$\sim 2 \cdot 10^{170}$

Note that all these quantities are far within the range of *real numbers* allowed on your *WP43* (see *App. B*). Physical constants are seldom known more precisely than twelve digits. Please take both facts into account when assessing very small numeric differences in your calculations as well as when talking about very large numbers (cf. f. 73).

Just *SI* base units and *SI* derived units are employed for all these constants, no prefixes. This eases their use in subsequent calculations.

System Flags

The *status bar*, especially its indicators (*SBI*), and the command STATUS return visible information about the system status of your *WP43* (cf. *OMS 2* and *5*). For program control, machine readable status information is available via 43 named *system flags* as listed below, sorted following their order of appearance in the *status bar*.

The status of *flags* with an orange *SBI* field is returned by STATUS – with a yellow field in short form only and in addition to other indication. If the third column is green, the corresponding *flag* may be modified by the user, else it can be written by the system only. *Flags* with their remarks printed on light blue are stored by STOCFG and recalled by RCLCFG. Some *system flags* can be also accessed via lettered *user flags*. *Startup default flag* status is indicated.

Purpose	*SBI*	*Flag*	Remarks
Date display	Date string	YMD, ~~DMY~~, ~~MDY~~	Indicating the date format chosen. The commands Y.MD, D.MY, and M.DY set the respective *flag* and clear the two others.
Time display	Time string	TDM24	**Set** for international 24h time display. Shows four digits for the year in the *status bar* date display.

[80] Almost everything is hydrogen there – you can neglect the other material on this scale.

Purpose	*SBI*	*Flag*	Remarks
			Clear for 12h time display: e.g., 1:23 will become 1:23am, 23:45 will become 11:45pm. Shortens the date display in the *status bar* to two digits for the year.
Complex results	ℂ / ℝ	~~CPXRES~~ (I)	**Set** for allowing *complex* results for *real* input (e.g., $\sqrt{-1}$). Else an error will be thrown in such cases.
Polar notation	⦿ / ⌙	~~POLAR~~ (X)	**Set** for polar display of *complex numbers*, **clear** for rectangular.
Fraction display 1	—	~~FRACT~~	**Set** for fraction display, **clear** for decimal display (a b/c sets FRACT, .d clears it). See next entries.
Fraction display 2	—	~~FRCSRN~~	**Set** for a verbose fraction display showing *register* names (e.g., x = 3/5), else clear.
Fraction kind 1	—	PROPFR	**Set** for *proper fractions*, **clear** for *improper fractions* (coming from decimal display, a b/c sets PROPFR, then it toggles it). PROPFR allows only *proper fractions* in display (e.g., 1 4/5 instead of 9/5); any *reals* (with $\|x\| < 10^6$) will be displayed as *proper fractions* according to the settings of DENANY, DENFIX, and D.MAX.
Fraction kind 2	see *OMS 2*	DENANY	**Set** if any denominator up to D.MAX is legal (D.MAX is indicated in the *status bar*).[81] For given D.MAX, this is the most precise way of displaying a decimal number as a fraction. See also next entry.
Fraction kind 3	see *OMS 2*	~~DENFIX~~[82]	**Set** if D.MAX is the one and only denominator allowed. **Clear** if the denominator may be an integer factor of D.MAX.[83]

[81] E.g., for D.MAX = 8 and DENANY, denominators 1, 2, 3, 4, 5, 6, 7, and 8 are allowed.

[82] DENFIX is evaluated only for ~~DENANY~~.

[83] If D.MAX = 60 and ~~DENFIX~~ (and ~~DENANY~~), this will allow for denominators 1, 2, 3, 4, 5, 6, 10, 12, 15, 20, 30, and 60. Note 60 was a sacred number in ancient *Babylon* – hence our hours are divided in 60 *minutes* even today. Actually, Babylonic astronomers,

Purpose	*SBI*	*Flag*	Remarks
Carry	c or C	~~CARRY~~ (C)	Reflects the status of the *Carry* bit. See *OMS 2*.
Overflow	o or C	~~OVERFL~~ (B)	Reflects the status of the *Overflow* bit. See *OMS 2*.
Leading zeros	—	~~LEAD.0~~ (L)	**Set** for leading zeros turned on in *short integers* of bases 2, 4, 8, and 16. [84]
Matrix grow mode	**grow** / **wrap**	~~GROW~~	**Set** (e.g., by M.GROW) if matrices may grow, else **clear** (e.g., by M.WRAP). Lit in the *status bar* as long as a matrix is open for editing. See J+ in the *IOI* and *OMS 2*. [85]
Payment due (for *TVM*)	**END** / **BEG**	ENDPMT	**Set** for payments at the end of each period, **clear** for beginning. Controlled by ENDP and BEGINP, cf. *OMS 5*. Lit in the *status bar* as long as TVM is open. [85]
Alpha	A or α	~~ALPHA~~	**Set** for *AIM*, else clear.
Upper case	A / α	αCAP	**Set** for capital letters, **clear** for lower case.
Running timer	⏱	~~RUNTIM~~	**Set** if the timer is running in background. Note your WP43 will not turn off automatically with the timer running – reliable power supply is recommended.
Running I/O	↕	~~RUNIO~~	**Set** if I/O is in progress.
Tracing	—	~~TRACE~~ (T)	**Set** if the 'print line' is in tracing mode. Else each print must be triggered explicitly.
User mode	U	~~USER~~	**Set** if your *WP43* is in user mode.
Low battery	🔋	~~LOWBAT~~	**Set** if battery voltage is low (< 2.5 V, cf. BATT? and *OMS 2*).
USB power	ψ	~~USB~~	**Set** if connected to a *USB* power source. [86]

businessmen, and computers calculated base 60.

[84] Works like *flag* 3 of *HP-16C*.

[85] Compare also GROW in the *HP-42S OM*, p. 213.

Purpose	*SBI*	*Flag*	Remarks
Fast function	—	~~FASTFN~~	If **set**, your *WP43* calculates faster without using internal float of extended (39-digit) precision.[86] This is a trade-off between speed and precision. It affects $\sqrt{x}$, $\sqrt[3]{x}$, ln, ln(1+x), lg, lb, $\log_x y$, e^x, e^x-1, 10^x, 2^x, y^x, sin, cos, tan, arcsin, arccos, arctan, sinh, cosh, tanh, arsinh, arcosh, and artanh.
Special results	—	~~SPCRES~~ (D)	**Set** for allowing 'special results' of calculations (i.e. ±∞ and NaN).
Stack size	8 / 4	SSIZE8	**Set** for 8, **clear** for 4 *stack registers*. Note *register* contents will remain unchanged if SSIZE8 is modified (also by RCLCFG).
Radix mark	—	DECIM.	**Set** for a decimal point, **clear** for a decimal comma.
Multiplication symbol	—	MULT×	**Set** for multiplication symbol ×, **clear** for ·. Note the symbols used for matrices and in equations are fixed.
Overflow from ALL	—	~~ALLENG~~ (A)	**Set** if *real numbers* exceeding the range displayable in ALL or FIX shall be shown in ENGineer's format, **clear** for SCIentific.
Automatic execution	—	~~AUTXEQ~~	With AUTXEQ set when you turn your *WP43* on (e.g., from previous session), AUTXEQ is cleared, and program execution begins at the current program line.[87]
Data entry	—	~~NUM.IN~~, ~~ALP.IN~~	**Set** for numeric or alphanumeric (text) entry, respectively.[88]
Error handling	—	~~IGN1ER~~	If **set**, your *WP43* ignores one arbitrary error and clears IGN1ER then.[89] The operation causing this error will not be executed.

[86] Speed of execution increases in following order and coarse relation: battery power supply and ~~FASTFN~~ (25), battery and FASTFN (70), *USB* power supply and ~~FASTFN~~ (80), *USB* and FASTFN (200). See also *App. A*.

[87] Works like *flag* 11 of *HP-42S*.

[88] These work like *flags* 22 and 23 of *HP-42S*.

Purpose	*SBI*	*Flag*	Remarks
Automatic OFF	—	AUTOFF	**Set** if automatic shutdown after 600 s is enabled. Else your *WP43* will remain ON until you will turn it off manually or battery voltage will drop below the lower limit (cf. BATT? and *OMS 2*).[90] Note *WP43* will not turn off automatically with a program or the timer running.
Beeper	—	~~QUIET~~	**Set** for disabled beeper.
Automatic *stack* lift	—	ASLIFT	**Cleared** by ENTER↑, CLX, M.EDI, M.EDIN, M.←, M.↑, M.→, M.↓, Σ+, and Σ− (see *App. H* and *OMS 1*). **Set** by all other functions.
Integrating	—	~~INTING~~	**Set** while the *Integrator* is running.
Solving	—	~~SOLVING~~	**Set** while the *Solver* is computing a root.
Variable menu	—	~~VMDISP~~	**Set** while the *variable menu* is displayed (cf. VARMNU and *OMS 3*).

Note the *status bar* will be updated every minute or after a keystroke. Thus, e.g., settings by SETDAT or SETTIM, although recorded immediately, may take up to one minute to appear in the *status bar* date-and-time string.

Tests

All commands provided under TEST are compiled and classified below for your convenience (*num. comp.* represents *numeric comparison*):

Name(s)	Legal input *data types*	Num. comp.	Remarks
CPX?	All	Yes	true for *DT* 3 or 9 and Im(x) ≠ 0
INT?	All	Yes	true for FP(x) = 0

[89] Works like *flag* 25 of *HP-42S.*

[90] Works like *flag* 44 of *HP-42S* so far.

Name(s)	Legal input *data types*	Num. comp.	Remarks
EVEN?, ODD?, PRIME?	All	Yes	**true** for FP(x) = 0 and … (cf. pp. 39, 68, and 71)
FP?	All	Yes	**true** for FP(x) ≠ 0
REAL?	All	Yes	**true** for *DT* 2 or 8 and FP(x) ≠ 0
MATR?	All	—	**true** for *DT* 8 or 9
M.SQR?	All	—	**true** if x is a square matrix
NaN?, SPEC?	All	Yes	**true** if x contains special values (cf. pp. 66 and 87)
STRI?	All	—	**true** for *DT* 7
x=?, x≠?	All	Yes	Cf. p. 98
x<?, x≤?, etc.	1,2,4,5,6,7,10	Yes	
x≈?	2,3,4,5,8,9	Yes	
x=+0?, x=-0?	1, 2, 10	Yes	
LEAP?	2, 6	Yes	Cf. p. 53
CONVG? ***s***	—	Yes	Cf. p. 31
ENTRY?	—	—	Tests the entry flag, cf. p. 39
KEY? ***d***	—	—	Tests for key press, cf. p. 51
LBL? ***label***	—	—	Does this *label* exist (cf. p. 53)?
TOP?	—	—	**true** in a top level routine, cf. p. 91

DRAFT

Nonprogrammable Commands and Keys

Operations whose names are printed on violet or black in the *IOI* can't be programmed (but *menu items* can be called). Also answering questions your *WP43* may ask is impossible automatically.

All other *items* may be used in programs. Operations triggered by P/R, α, ↶, ≡△, ▲, ≡▽, and ▼, however, may not be echoed on the screen (neither in *RUM* nor *PEM*) as the other commands are.

SECTION 2: MENUS AND CATALOGS

Due to the many *items* your *WP43* features, most of them are stored in *menus* (as introduced in *OMS 1*). Besides commands, also numeric constants, *characters* (as in the alpha *menus*), digits, *submenus*, *variables*, and program *labels* may be contained in *menus*.

If no specific *menu* was called by you, either MyMENU or (in *PEM*) MyPFN or (in *AIM*) Myα are displayed.

You may switch *menus* (except *catalogs* – see below) easily just calling another *menu* directly from the *menu* you are using – no need to EXIT first. The last (*sub*-) *menus* opened are stored on a 4-level *menu stack*. Each EXIT of a (sub-) *menu* pops the youngest entry from this *stack*; opening a new (*sub*-) *menu* pushes this new entry on it. So wherever you are, the fourth EXIT will return to a screen with MyMENU, MyPFN, or Myα displayed at latest.

Catalogs are a special kind of *menus*: their contents are sorted alphabetically. Your *WP43* provides 17 *catalogs*:

- CATALOG'FCNS (by far the largest *catalog* in *SDC*),
- CATALOG'MENUS ,
- CATALOG'CHARS'αINTL ,
- the two *submenus* of CATALOG'PROGS ,
- the 11 *submenus* of CATALOG'VARS , and
- CONST (being identical to CATALOG'CONST).

Let's look at the mother of all *catalogs* first.

One to Find and Rule Them All – the CATALOG

CATALOG opens a very special *menu*: CATALOG comprises all the visible *items* defined on your *WP43*. Many of them are sorted alphabetically: these we call *cataloged*. Individual *cataloged items* may be accessed quickly as demonstrated in next chapter.

DRAFT

A survey of the various CATALOG branches at *startup* is presented below. Note they appear here in reverse order compared to the *LCD*, taking care of common top-down reading habits:

	F1	F2	F3	F4	F5	F6	Remarks
CATALOG:	FCNS	CONST	CHARS	PROGS	VARS	MENUS	top branches

FCNS:	°C→°F	°F→°C	10^x	1COMPL	$1/x$	2COMPL	*catalog* of all the over 700 functions provided; while in *AIM*, only ← and → will be shown
	2^x	$\sqrt[3]{x}$	ABS	ACOS	ActUSB	ac→ha	
	ac_{US}→ha	AGM	AGRAPH	ALL	AND	arccos	
	…						
	…	…	#B				
CONST:	a	a_0	…				constants, cf. pp. 113ff

CHARS:	αINTL	A…Ω		αMATH	Myα	α·	*character* branches
αINTL:	A	À	Á	Â	Ã	Ä	*catalog* of all the 99 international Latin letters provided, s. p. 147
	…						
	…	…	Ž				
αMATH:	<	≤	=	≈	≥	>	mathematical operators and symbols, see p. 148
	…						
A…Ω:	A	B	Γ	Δ	E	Z	Greek letters, see p. 148
	…						
α•:	!	;	:	‘	“	√	punctuation marks etc., see p. 149
	…						
PROGS:							*global labels* defined (empty unless `test_pgms.bin` was copied); to be filled with your creations

VARS:	L.INTS	S.INTS	REALS	CPXS	ALL	MATRS	branches for various types of *variables*
	DATES	TIMES	ANGLES	CONFIG		STRING	
ALL:	ACC	ADM	D.MAX	FV	GRAMOD	ISM	*catalogs* of all *variables* currently specified, placed according to their *DT*s – most of
	i%/a	Mat_A	Mat_B	Mat_X	NPER	PER/a	
	PMT	PV	REALDF	↑Lim	↓Lim	#DEC	

DRAFT

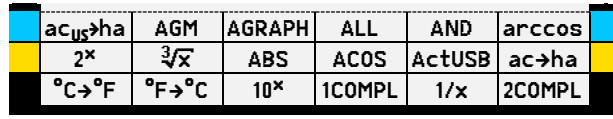
	F1	F2	F3	F4	F5	F6	Remarks
ANGLES:							these *submenus* are empty in *SDC*. They will be filled and grow with your creations. ALL is the union of all the other *submenus* under VARS
CPXS:							
CONFIG:							
DATES:							
L.INTS:	ADM	D.MAX	GRAMOD	ISM	REALDF	#DEC	
MATRS:	Mat.A	Mat.B	Mat.X				
REALS:	ACC	FV	i%/a	NPER	PER/a	PMT	
	PV	↑Lim	↓Lim				
STRING:							
S.INTS							
TIMES:							

MENUS:	ADV	ANGLES	A:	Binom:	BIT	Cauch:	*(sub-) menus* currently defined – see above and below for fix *menu* contents
	CHARS	CLK	CLMy...	CLR	CONST	CPX	
	CPXS	DATES	DISP	EQN	EXP	Expon:	
	...						
	...	↺→					
ADV:	…						*(sub-) menus* provided unless mentioned above already, see pp. 138ff for fix contents. Your creations will be inserted as new entries here
A:	…						
Binom:	…						
BIT:	…						
…							

The branches FCNS, CONST, and CHARS are fixed since all functions, constants, and *characters* on your *WP43* are predefined; MENUS, PROGS, and the *submenus* of VARS are expandable since you may create *items* of these kinds (cf. *OMS 6*).

Opening CATALOG will display these six top level labels, pointing to the *submenus* containing all the functions, constants, *characters*, programs, *variables*, and *menus* defined at execution time. Choosing one of these *submenus* will display its first *view* of *items* (primary,

🟧- and 🟦-shifted, as applicable). Pressing FCNS, for instance, will call CATALOG'FCNS showing its first *view* as pictured here:



Select any *function* by pressing the corresponding *softkey* in this *catalog*. You can as well choose any *menu* via CATALOG'MENUS, an arbitrary *real variable* via CATALOG'VARS'REALS, etc.

Within any *catalog*, ▲ will advance just by six *items* per keystroke (and ▼ will go back by six); but you may as well use the alphabetic search method explained in next chapter. EXIT will just leave the open branch without doing anything.

Although all the functions provided on your *WP43* are stored in CATALOG'FCNS, most of them can be accessed easier through other *menus* (see the chapter after next). Remember that each and every command on your *WP43*, as well as the keystrokes calling it are explained for your reference in the *IOI* on pp. 20ff; also the predefined *menus* and *variables*, as well as the *system flags* are listed therein. The constants provided are printed on pp. 113ff.

Accessing Cataloged Items Rapidly

You can browse a *catalog* like any other *menu* just using ▲ and ▼. In CONST and major parts of CATALOG (i.e. FCNS, CONST, MENUS, PROGS, CHARS αINTL, and the *submenus* of VARS), however, you may reach your target a good deal faster taking advantage of the *alphabetic search method* demonstrated below.[91]

Example:

Let's assume you are looking for the function NEXTP but have forgotten in which particular *menu* it is stored. Then simply enter:

[91] This method is known from *WP 34S*.

1	User input	CATALOG FCNS
	Output	Your *WP43* displays the first *view* in this *catalog*;[92] *AIM* is on.

ac_{US}→ha	AGM	AGRAPH	ALL	AND	arccos
2^x	$\sqrt[3]{x}$	ABS	ActUSB	ACOS	ac→ha
°C→°F	°F→°C	10^x	1COMPL	1/x	2COMPL

2	User input	1st *character* of the *item* desired e.g., N
	Output	Your *WP43* displays a new *view* starting with the first *item* starting with this *character* [93]

NOR	$Norml_p$	$Norml_{\triangle}$	$Norml_{\triangle}$	$Norml^{-1}$	NOT
NEIGHB	NEXTP	nmi→km	nmi→m	Nm→lbft	NOP
NAND	NaN?	$NBin_p$	$NBin_{\triangle}$	$NBin_{\triangle}$	$NBin^{-1}$

3	User input	Here it is already! Press the corresponding *softkey* to call it e.g., f F2. [94]
	Output	If no parameter input is required, your *WP43* jumps directly to step 4. Else it turns to *TAM* and waits for your input:
3a	User input	Enter the necessary parameter(s).

[92] … unless you visited the same *catalog* before – then it will open showing its last view you looked at. The remaining procedure will stay unchanged though.

[93] This search is case independent (i.e. specifying **A** will find **a** as well). Nevertheless **A** and **a** remain different letters. You can search for Greek letters via prefix f, e.g., f + A for α (watch the sorting order as printed at begin of *Section 1*). Also other *characters* can be specified in a search – cf. the *virtual keyboard* printed on p. 166. Note that the *items* in the *catalog* you search may be shown at other positions in the *menu section* on display than the ones you see browsing using ▼ or ▲ only.

[94] You may enter more *characters* – but after 3 *seconds* or after pressing ▼ or ▲, whatever comes first, the search string will be reset. Then you may continue browsing using ▼ or ▲, or enter a new search string.

If the *character* or string specified is not found in this catalog, however, then the first *item* following in sorting order (cf. begin of *Section 1*) will be shown. If there is no such *item* then the last *item* in this *catalog* will be displayed.

DRAFT

4	Output	Your *WP43* executes the command, calls the program, recalls the constant, *RoV*, inserts the command, digit or *character* selected. It displays the result then, e.g., 1 237 572 153 053 … and exits this *catalog* returning to the mode set before entering it.

At the bottom line, this means that ...

- you can call an arbitrary function (out of over 700) by CATALOG FCNS + 4 keystrokes maximum if you know its first two *characters*;
- you can recall any constant provided CONST + 3 keystrokes max. if you know its first *character*;
- you can insert any Latin letter provided by CATALOG CHARS αINTL (or in *AIM* by f A) + 3 keystrokes max.

αINTL is the one and only *catalog* staying open until you EXIT it explicitly. Picking an *item* from any other *catalog* (e.g. from CONST) will EXIT this automatically.

Further Menus and Their Contents

In the table following, all the 38 *menus* provided directly accessible from the keyboard (incl. those accessible in *AIM* and three user *menus*) are listed in alphabetical order with their number of *views* and *submenus* contained:

Menu	*Views*	*Submenus*	*Views*	Remarks
ADV	1	∫fdx	1	
BIT	3	—		
CLK	2	—		
CLR	1	—		

Menu	*Views*	*Submenus*	*Views*	Remarks
(CATALOG') CONST	5	—		cf. pp. 113ff for the content of this *catalog*
CPX	1	—		
DISP	2	—		
EQN	1	5	1 each	
EXP	1	—		
FIN	1	TVM	1	
FIT	2	ASSESS, HISTOG, PLOT	1 each	
FLAG	1	—		
INFO	2	—		
INTS	3	—		
I/O	1	—		
LOOP	1	—		
MODE	1	—		
MATX	2	M.EDIT, M.SIMQ	1 each	
MyMENU	1	—	Three *menus* to be filled by you – feel free to create more (cf. *OMS 6*)	
MyPFN	1	—		
Myα	1	—		
PART	1	—		
PRINT	1	—		
PROB	2	14	1 each	Probability distributions
P.FN	1	P.FN2	1	
STAT	2	FIT and HISTOG	2	Cf. FIT above
STK	1	—		
TEST	2	—		
TRI	1	—		

DRAFT

Menu	*Views*	*Submenus*	*Views*	Remarks
U→	2	8	1 - 3	Conversion kinds
X.FN	2	Ellipt and Orthog	1 each	Function kinds
αINTL A	6	—		*Submenu* of CATALOG'CHARS
αMATH ≈	3	—		
α.FN	1	—		
A…Ω	2	—		*Submenu* of CATALOG'CHARS
α●	3	—		
Σ	2	—		
∢→	2	—		

Below, these *menus* are listed in full detail, usually like you see them on display. Different *views* within the same *menu* are separated by dashed lines, *submenus* by double lines in this table. Note that individual *tags* or *labels* may appear in more than one *menu* and also on the keyboard.

Menu	F1	F2	F3	F4	F5	F6	Remarks
ADV							advanced operations, see *OMS 4*
	PGMSLV	SLVC	f"(x)			PGMINT	
	SOLVE	SLVQ	f'(x)	Π_n	Σ_n	∫fdx	
∫fdx		Draw	ACC	↓Lim	↑Lim	∫	
BIT	SB	BS?	#B	FB	BC?	CB	contains all the *Boole's* and bit operations (first two views) and settings (third view) of *HP-16C* and *WP 34S*
	NAND	NOR	XNOR	MIRROR			
	AND	OR	XOR	NOT	MASKL	MASKR	
	LJ				RJ	ASR	
	SL	RL	RLC	RRC	RR	SR	
	A	B	C	D	E	F	
	1COMPL	2COMPL	UNSIGN	SIGNMT		WSIZE	

DRAFT

Menu	F1	F2	F3	F4	F5	F6	Remarks
CLK	J→D	D→J		DAY	MONTH	YEAR	date and time functions (first view) and settings (2nd view)
	DATE	→DATE	DATE→	WDAY	TIME	x→DATE	
	J/G						
	SETTIM	TDISP	SETDAT	D.MY	Y.MD	M.DY	
CLR	CLall	CLBKUP	DelVall		DelMall	CLLCD	commands for clearing and deleting
	CLREGS	CLPall	CLVall	CLFall	CLMall	CLSTK	
	CLΣ	CLP	CLCVAR	CF	CLMy...	CLX	
	RESET		CLKEYS	DelITM	CLMENU		
CLMy...				CLMyM	CLMyP	CLMyα	
CONST		*catalog* of constants					see pp. 113ff
CPX							*complex* functions
	CX→RE	RE→CX		Im			
	dot	cross	UNITV	Re	conj	Re⇄Im	
DISP							display rounding and shifts, formats and settings, mostly for *reals*
	SDL	SDR			RDP	RSD	
	FIX	SCI	ENG	ALL	ROUNDI	ROUND	
	GAP	HIDE	HIDE?	RANGE	RANGE?	DSTACK	
	CHINA	EUROPE	INDIA	JAPAN	UK	USA	
EQN							equations (see *OMS 4*)
	DELETE						
	NEW	EDIT	f''	f'	∫f	Solver	
EQ.EDI	←	( )	^	:	=	→	Equation Editor
Solver					Draw	Calc	show the *names* of all *variables* in the current equation and more
∫f						Draw	
f'							
f''							
EXP	sinh	arsinh	cosh	arcosh	tanh	artanh	exponential, logarithmic, and hyperbolic functions
	$\sqrt[3]{x}$	$\sqrt[x]{y}$		ln 1+x	e^x-1		
	x^3	y^x	$\log_x y$	lb x	2^x	x^2	

Menu	F1	F2	F3	F4	F5	F6	Remarks
FIN	%	%MRR	%T	%Σ	%+MG	TVM	financial functions and settings (cf. *OMS 5*). 2nd row is for recalling only
	Begin					End	
	RCL n_P	i%/a	per/a	PV	PMT	RCL FV	
TVM	n_{PER}	i%/a	per/a	PV	PMT	FV	
	CLΣ	ASSESS				PLOT	*2D* sample statistics, regression, and *MSA*. See p. 145 for HISTOG. The 2nd *view* is for selecting curve fit model(s)
	Σ-		s_{xy}	cov		HISTOG	
FIT	Σ+	L.R.	s(a)	r	$\hat{x}$	$\hat{y}$	
	GaussF	CauchF		BestF			
	ParabF	HypF	RootF				
	LinF	ExpF	LogF	PowerF		OrthoF	
ASSESS	NXTFIT	ZOOM					see *OMS 2*
PLOT	CENTRL	s_{mi}					
						CLFall	
	FS?S	FS?C	FS?F	FC?F	FC?S	FC?C	
FLAG	SF	FS?	FF	STATUS	FC?	CF	
	WHO?	VERS?	DIM?	±∞?	αPOS?	αLENG?	system information plus one non-binary test (±∞?)
	LocR?	DISK?	ULP?	NEIGHB	SDIGS?	BATT?	
INFO	SSIZE?	MEM?	RM?	ISM?	WSIZE?	KTYP?	
	RANGE?	HIDE?	J/G?	VOL?		BestF?	
	DBL/	DBLR	DBL×	^MOD	ceil	GCD	digits for *short integers* with bases > 10, *integer* operations
	IDIV	RMD	MOD	×MOD	floor	LCM	
INTS	A	B	C	D	E	F	
	1COMPL	2COMPL	UNSIGN	SIGNMT		WSIZE	
	→BIN	→DEC			→OCT	→HEX	
	BEEP	TONE	BUZZ	PLAY	VOL-	VOL+	backup, data exchange and signalling
	LoadSS	LoadV	LoadR	LoadΣ	READP		
I/O	LoadAll	LoadPr	LoadEq	DUMP	WRITEP	SAVEST	

DRAFT

Menu	F1	F2	F3	F4	F5	F6	Remarks
	DEC					INC	
LOOP	DSE	DSZ	DSL	ISE	ISZ	ISG	
	ENORM		STOEL	RCLEL	PUTM	GETM	matrix operations (*items* are sorted almost as in *HP-42S*)
	dot	cross	UNITV	DIM	INDEX	EDITN	
MATX	NEW	$[M]^{-1}$	|M|	$[M]^T$	SIM EQ	EDIT	
	CMAX	CMIN	M.QR	FIND	EIGVAL	EIGVEC	
	RNORM	RSUM	M.LU	DIM?		R⇄R	
	I+	I-	STOIJ	RCLIJ	J-	J+	
	INSR		DELR		WRAP	GROW	*Matrix Editor* (as in *HP-42S*)
M.EDIT	←	↑	OLD	GOTO	↓	→	
M.SIMQ	Mat A	Mat B				Mat X	for systems of linear equations
	DMCP						mode settings; **DMCP** calls *SM*'s *DMCP* menu on the calculator
		GRAD	RM		SETSIG	DENMAX	
MODE	SF	DEG	RAD	MIL	MULπ	CF	
							will pop up in *RUM* beyond *AIM* [95]
MyMENU							
							will pop up in *PEM* beyond *AIM* [95]
MyPFN	LOOP	TEST			P.FN	P.FN2	
							will pop up in *AIM* [95]
Myα	←	αMATH	α.	A...Ω	αINTL	→	
					Re	Im	
PART	IP	FP	sign	MANT	EXPT	DECOMP	

[95] ... as long as no other *menu* is called (see *OMS 6*). MyMENU is empty at startup. All three My… *menus* can be filled and edited by you.

Menu	F1	F2	F3	F4	F5	F6	Remarks
PRINT (⇧)	🖶x	🖶r	🖶Σ	🖶ADV		🖶PROG	like *WP 34S* and similar to PRINT commands of *HP-42S*
	🖶STK	🖶REGS	🖶USER	🖶TAB	🖶#	🖶CHAR	
				🖶WIDTH	🖶DLAY	🖶MODE	

		NBin:	Geom:	Hyper:	Binom:	Poiss:	combinations, permutations, random number generators, and 14 probability distributions. Selecting one (e.g., NORML) opens a *sub-menu* featuring four entries for its *PDF* (or *PMF*), its *CDF*, error probability, and quantile function
	LgNrm:	Cauch:		Expon:	Logis:	Weibl:	
PROB	Norml:	t:	C_{yx}	P_{yx}	F:	χ^2:	
	RAN#	SEED	RANI#		lnΓ	Γ(x)	
Binom:	$Binom_p$		$Binom_{▲}$	$Binom_{▲}$		$Binom^{-1}$	
Cauch:	$Cauch_p$		$Cauch_{▲}$	$Cauch_{▲}$		$Cauch^{-1}$	
Expon:	$Expon_p$		$Expon_{▲}$	$Expon_{▲}$		$Expon^{-1}$	
F:	$F_p(x)$		$F_{▲}(x)$	$F_{▲}(x)$		$F^{-1}(p)$	
Geom:	$Geom_p$		$Geom_{▲}$	$Geom_{▲}$		$Geom^{-1}$	
Hyper:	$Hyper_p$		$Hyper_{▲}$	$Hyper_{▲}$		$Hyper^{-1}$	
LgNrm:	$LgNrm_p$		$LgNrm_{▲}$	$LgNrm_{▲}$		$LgNrm^{-1}$	
Logis:	$Logis_p$		$Logis_{▲}$	$Logis_{▲}$		$Logis^{-1}$	
NBin:	$NBin_p$		$NBin_{▲}$	$NBin_{▲}$		$NBin^{-1}$	
Norml:	$Norml_p$		$Norml_{▲}$	$Norml_{▲}$		$Norml^{-1}$	
Poiss:	$Poiss_p$		$Poiss_{▲}$	$Poiss_{▲}$		$Poiss^{-1}$	
t:	$t_p(x)$		$t_{▲}(x)$	$t_{▲}(x)$		$t^{-1}(p)$	
Weibl:	$Weibl_p$		$Weibl_{▲}$	$Weibl_{▲}$		$Weibl^{-1}$	
χ^2:	$\chi^2{}_p(x)$		$\chi^2{}_{▲}(x)$	$\chi^2{}_{▲}(x)$		$(\chi^2)^{-1}$	

	PUTK	PROMPT	AVIEW	CNST	TICKS	REM	additional programming functions (avoided a *multi-view menu* here to allow for easier program browsing)
	R-CLR	R-COPY	R-SORT	R-SWAP	LocR	PopLR	
P.FN	INPUT	MVAR	VarMNU	END	PAUSE	P.FN2	
	AGRAPH	PIXEL	POINT	ERR	MSG	XEQ.SKP	
	BACK	CASE	SKIP	NOP	SDL	SDR	
P.FN2	MENU	KEYG	KEYX	CLMENU	EXITall	RTN.SKP	

DRAFT

Menu	F1	F2	F3	F4	F5	F6	Remarks
STAT	CLΣ	$\bar{x}_G$	ε	ε_p	ε_m	FIT	*1D* and *2D* statistics of independent samples – cf. also FIT on p. 142
	Σ-	$\bar{x}_w$	s_w	σ_w	s_{mw}	HISTOG	
	Σ+	$\bar{x}$	s	σ	s_m	SUM	
		$\bar{x}_H$					
		$\bar{x}_{RMS}$	x_{min}	x_{max}	Δx		
		MEDIAN	MAD	Q_3-Q_1	%ile		
HISTOG	HISTOX	HISTOY	↓BIN	nBINS	↑BIN	HPLOT	see *OMS 2*
HPLOT	HNORM						
STK							*stack* operations
	x⇄	y⇄	z⇄	t⇄	⇄	DROPy	
TEST	ENTRY?	KEY?	LBL?	STRI?	CONVG?	TOP?	binary tests.
	INT?	EVEN?	ODD?	PRIME?	LEAP?	FP?	
	x<?	x≤?	x=?	x≠?	x≥?	x>?	
	SPEC?	NaN?		M.SQR?			
	x=+0?	x=-0?	x≈?	MATR?	CPX?	REAL?	
TRI	sinh	arsinh	cosh	arcosh	tanh	artanh	trigonometric & hyperbolic functions (cf. EXP).
	sinc	sincπ				atan2	
	sin	arcsin	cos	arccos	tan	arctan	
U→							unit conversions (see pp. 150ff for more).
	°C ←	→ °F			V:	A:	
	E:	P:	Time:	F&p:	m:	x:	
	dB ←→	power ratio	dB ←→	field ratio	Nm ←→	lbf·ft	
A:	ha ←→	acre	m^2 ←→	ha	ha ←→	$acre_{US}$	units of area "kitchen units" (see pp. 152ff)
	m^2 ←→	acre	m^2 ←→	mŭ	m^2 ←→	$acre_{US}$	
E:	J ←→	cal	J ←→	Btu	J ←→	Wh	units of energy
F&p:			Pa ←→	mmHg			units of force and pressure
	Pa ←→	in.Hg	Pa ←→	torr	Pa ←→	atm	
	N ←→	lbf	Pa ←→	bar	Pa ←→	psi	

DRAFT

Menu	F1	F2	F3	F4	F5	F6	Remarks
	kg ↔	→stone	kg ↔	→short cwt	kg ↔	troy oz.	units of mass
m:	kg ↔	lb.	kg ↔	cwt	kg ↔	oz.	
	kg ↔	liǎng			kg ↔	jīn	
	kg ↔	ton	kg ←	→short ton	kg ↔	→carat	
	g ↔	oz.	g ↔	troy oz.	g ↔	→carat	kitchen units
P:	W ↔	hp_E	W ↔	hp_{UK}	W ↔	hp_M	units of power
Time:	s ↔	day			s ↔	year	units of time
	m^3 ↔	→$floz_{UK}$			m^3 ↔	→$floz_{US}$	units of volume
V:	m^3 ↔	gal_{UK}	m^3 ↔	→barrel	m^3 ↔	gal_{US}	
	mℓ ↔	$floz_{UK}$	mℓ ↔	$pint_{UK}$	ℓ ↔	gal_{UK}	*Imperial* kitchen units
	mℓ ↔	tsp_{UK}	mℓ ↔	$tbsp_{UK}$	mℓ ↔	cup_{UK}	
	mℓ ↔	tea $spoon_I$	mℓ ↔	table $spoon_I$	mℓ ↔	cup_I	international kitchen units
			ℓ ↔	quart	ℓ ↔	gal_{US}	*US* customary kitchen units
	mℓ ↔	$floz_{US}$	mℓ ↔	$pint_{US}$			
	mℓ ↔	tsp_{US}	mℓ ↔	$Tbsp_{US}$	mℓ ↔	cup_{US}	
	m ↔	inch	m ↔	point	m ↔	yard	units of length
	m ↔	mile	m ↔	nmi.	m ↔	foot	
x:	m ↔	au	m ↔	l.y.	m ↔	pc	
	m ↔	→fathm					
	m ↔	chǐ	m ↔	cùn	m ↔	fēn	
	m ↔	lǐ	m ←	→ yǐn	m ↔	→zhàng	
	km ↔	mile	km ↔	nmi.	m ↔	→surv. $foot_{US}$	kitchen units
	mm ↔	inch	mm ↔	point			

	$J_y(x)$	lnβ	lnΓ	max	min	NEXTP	advanced mathematical functions like Beta, Bessel, etc.
	erfc	FACTOR	FIB	g_d	g_d^{-1}	I_{xyz}	
X.FN	AGM	B_n	B_n^*	erf	Ellipt	Orthog	
	Γ_q	γ_{xy}	Γ_{xy}	ζ(x)	$(-1)^x$		
	W_m	W_p	W^{-1}	$Y_y(x)$	β(x,y)	Γ_p	

Menu	F1	F2	F3	F4	F5	F6	Remarks
Ellipt	ψ(υ,m)			F(φ,m)	E(φ,m)	Z(φ,m)	elliptic functions and integrals
	sn	cn	dn	K(m)	E(m)	Π(n,m)	
Orthog	H_{nP}						orthogonal polynomials
	H_n	L_m	$L_{m\alpha}$	P_n	T_n	U_n	

αINTL (A)											[α] *catalog* of all Latin letters provided, printed here in reverse vertical order following common reading habits. All letters (except ß) will change when case is switched in *AIM* – note you will see the individual letters displayed in either case only at one time. You can reach each of these 99 letters by ≤ 3 keystrokes via alphabetical access as explained on pp. 136f. You might be even faster using your computer keyboard in the *Simulator*.
A	a	À	à	Á	á	Â	â	Ã	ã	Ä	ä	
Å	å	Æ	æ	Ā	ā	Ă	ă	Ą	ą	B	b	
C	c	Ç	ç	Ć	ć	Č	č	D	d	Đ	ð	
Ď	ď	Đ	đ	E	e	È	è	É	é	Ê	ê	
Ë	ë	Ē	ē	Ĕ	ĕ	Ė	ė	Ę	ę	Ě	ě	
F	f	G	g	Ğ	ğ	H	h	I	i	Ì	ì	
Í	í	Î	î	Ï	ï	Ī	ī	Ĭ	ĭ	Į	į	
İ	ı	J	j	K	k	L	l	Ĺ	í	Ľ	ľ	
Ł	ł	M	m	N	n	Ñ	ñ	Ń	ń	Ň	ň	
O	o	Ò	ò	Ó	ó	Ô	ô	Õ	õ	Ö	ö	
Ø	ø	Ō	ō	Ŏ	ŏ	Œ	œ	P	p	Q	q	
R	r	Ŕ	ŕ	Ř	ř	S	s	Ś	ś	Ş	ş	
Š	š		ß	T	t	Ţ	ţ	Ť	ť	U	u	
Ù	ù	Ú	ú	Û	û	Ü	ü	Ũ	ũ	Ū	ū	
Ŭ	ŭ	Ů	ů	Ų	ų	V	v	W	w	Ŵ	ŵ	
X	x	Y	y	Ý	ý	Ŷ	ŷ	Ÿ	ÿ	Z	z	
Ź	ź	Ż	ż	Ž	ž							

Menu	F1	F2	F3	F4	F5	F6	Remarks
αMATH (≈)	·/× [96]	÷//[97]	∫	∞	∞	∞	[α] for inserting comparison symbols, parentheses & brackets, as well as more mathematical and related symbols. You can reach every *character* herein by ≤ 3 keystrokes.
	{	[	(	)	]	}	
	<	≦	=	≈	≧	>	
	$\bar{x}$	$\bar{y}$	$\hat{x}$	$\hat{y}$	$\overline{x}$	$\overline{y}$	
	∤	⊾	⊥	∛	√	∜	
	¬	∧	∨	≠	|	&	
	±	^	⊤	-1	ℏ		
	⊙	⊛	⊕	ℂ	ℝ		
	:=	≙	≡	«	»	ε	
α.FN	FBR						dedicated operations on *text strings* plus the font browser
					αLENG?	αPOS?	
	x→α	αRL	αRR	αSL	αSR	α→x	

A…Ω													[α] Greek letters. The keyboard grants direct access to 24 of them. [98] Note the two kinds of lower case Σ. Cf. αINTL for more.
	Α	α	Β	β	Γ	γ	Δ	δ	Ε	ε	Ζ	ζ	
	Η	η	Θ	ϑ	Ι	ι	Κ	κ	Λ	λ	Μ	μ	
	Ν	ν	Ξ	ξ	Ο	ο	Π	π	Ρ	ρ	Σ	ϲ	
		ς	Τ	τ	Υ	υ	Φ	φ	Χ	χ	Ψ	ψ	
	Ω	ω		ά		έ		ή		ί		ΰ	
	Ϊ	ϊ		ό		ύ	Ÿ	ϋ		ΰ		ώ	

[96] With *SDC*, the multiplication dot is found here and the cross is called via ■(x) in *AIM*. For ~~MULT×~~, this dot is called via ■(x) in *AIM* and the cross via (≈).

[97] With *SDC*, the obelus is found here and the solidus is called via ■(/) in *AIM*. For ~~MULT×~~, the obelus is called via ■(/) in *AIM* and the solidus via (≈). An unshifted division colon is in α●.

[98] The classical Greek alphabet goes **a**lpha, **b**eta, **g**amma, **d**elta, **é**-psilon, **z**ēta, **ē**ta, **th**ēta, **i**ōta, **k**appa, **l**ambda, **m**ȳ, **n**ȳ, **x**ī, **ó**-mikron, **p**ī, **rh**ō, **s**igma, **t**au, **ý**-psilon, **ph**ī, **ch**ī, **ps**ī, and **ō**-mega (ancient Greek letters H, Θ, and Y are taught to be pronounced like Finnish ÄÄ, T, and Y – Finnish Y sounds like French U or German Ü, unlike English U!). Think of *Nils Holgersson's* goose *Yksi* (followed by *Kaksi*, *Kolme*, *Neljä*, *Viisi*, and *Kuusi* for obvious reasons).

Greek pronunciation changed significantly in 2500 years: H, I, and Y all sound like I now.

Menu	F1	F2	F3	F4	F5	F6	Remarks
α● (⊡)	$	€	%	&	£	¥	[α] for inserting punctuation marks, currency symbols, arrows, and further special *characters*.
	¡	¿	§	ø	~	\	
	!	;	:	'	"	√	
	⌛	⚽	U	🔋	@	*	
	«	»		🕓	⏱	*	
	←	↑	↕	⇄	↓	→	
	,,	”	...	_	•		
Σ	Σx^2y	$\Sigma x\ln y$	$\Sigma x^2\ln y$	$\Sigma \ln y/x$		$\Sigma y\ln x$	all the sums necessary and accumulated for the statistics in STAT.
		$\Sigma \ln x$	$\Sigma \ln^2 x$	$\Sigma \ln xy$	$\Sigma \ln^2 y$	$\Sigma \ln y$	
	n	Σx	Σx^2	Σxy	Σy^2	Σy	
	Σx^3	Σx^4					
	$\Sigma x^2/y$	Σ^1/x	Σ^1/x^2	$\Sigma x/y$	Σ^1/y^2	Σ^1/y	
∡→	D→R	R→D	Mπ→R	R→Mπ	D→D.MS	D.MS→D	angular conversions, cf. pp. 161f.
	DEG→	RAD→	MULπ→	MIL→	GRAD→	D.MS→	
	→DEG	→RAD	→MULπ	→MIL	→GRAD	→D.MS	
	D→MIL	MIL→D	MIL→R	R→MIL			

The *menus* below will pop up after particular commands. Their *sub-menus* VAR and SYS.FL will display only writeable *items* for write operations but all for read. These *menus* will appear when calling…

	F1	F2	F3	F4	F5	F6	Remarks
… STO and RCL	...EL	...IJ					
	Config	Stack			min	max	
	→	VAR	X	Y	Z	T	see *OMS 1*
… x⇄, y⇄, etc.	→	VAR	X	Y	Z	T	see *OMS 1*
… x<?, etc.	0.	1.					
	→	VAR	X	Y	Z	T	see *OMS 1*

DRAFT

	F1	F2	F3	F4	F5	F6	Remarks
… SF, CF, etc.	→	SYS.FL	X	Y	Z	T	see *OMS 1*
… XEQ, GTO, etc.	→	PROG	X	Y	Z	T	see *OMS 3*
… DELITM				PROGS	VARS	MENUS	see *OMS 6*

PROG will display all *global labels* present in *RAM* at execution time. PROGS, VARS, and MENUS are identical to the respective CATALOG *submenus* (cf. pp. 133ff).

Unit Conversions

Your *WP43* features 22 angular conversions provided in ∠→ (cf. p. 149) and 142 unit conversions in U→ (cf. pp. 145f). The latter *menu* is subdivided into various branches as shown above. If local units will be dropped once for *SI*, just 18 conversions will do beyond the angular ones.

The seven fundamental *SI* units are *kelvin*, *meter*, *kilogram*, *second*, *ampere*, *mol*, and *candela*.[99] Beyond these and products, powers, and ratios of them, knowing the symbols and names of some *SI derived* (a.k.a. *compound*) *units* is helpful:

Quantity	Unit	Symbol and formula
Temperature	*degree Celsius*	$\vartheta[°C] = T[K] - 273.15$
Force	*newton*	$1\,N = 1\,kg\ {}^{m}/_{s^2}$
Pressure	*pascal*	$1\,Pa = 1\ {}^{N}/_{m^2} = 1\ {}^{kg}/_{m\,s^2}$
Energy	*joule*	$1\,J = 1\,N\,m = 1\,kg\ {}^{m^2}/_{s^2}$

[99] *Candela* measures *luminous intensity*.

TNG: *Luminous intensity = Lichtstärke*. Umseitig bedeuten *luminous flux = Lichtfluss* oder *Lichtstrom* und *luminance = Leuchtdichte*.

DRAFT

Quantity	Unit	Symbol and formula
Power	*watt*	$1\,\mathrm{W} = 1\ {}^{\mathrm{J}}/_{\mathrm{s}}$
Electric potential, voltage	*volt*	$1\,\mathrm{V} = 1\ {}^{\mathrm{W}}/_{\mathrm{A}} = 1\ {}^{\mathrm{J}}/_{\mathrm{A\,s}}$
Charge	*coulomb*	$1\,\mathrm{C} = 1\,\mathrm{A\,s}$
Capacitance	*farad*	$1\,\mathrm{F} = 1\ {}^{\mathrm{C}}/_{\mathrm{V}} = 1\ {}^{\mathrm{A\,s}}/_{\mathrm{V}}$
Resistance	*ohm*	$1\,\Omega = 1\ {}^{\mathrm{V}}/_{\mathrm{A}}$
Conductance	*siemens*	$1\,\mathrm{S} = 1\ {}^{\mathrm{A}}/_{\mathrm{V}}$
Magnetic flux	*weber*	$1\,\mathrm{Wb} = 1\,\mathrm{V\,s}$
Magnetic flux density	*tesla*	$1\,\mathrm{T} = 1\ {}^{\mathrm{Wb}}/_{\mathrm{m^2}} = 1\ {}^{\mathrm{V\,s}}/_{\mathrm{m^2}}$
Inductance	*henry*	$1\,\mathrm{H} = 1\ {}^{\mathrm{Wb}}/_{\mathrm{A}} = 1\ {}^{\mathrm{V\,s}}/_{\mathrm{A}}$
Frequency	*hertz*	$1\,\mathrm{Hz} = {}^{1}/_{\mathrm{s}}$
Radioactivity	*becquerel*	$1\,\mathrm{Bq} = {}^{1\,\mathrm{decay}}/_{\mathrm{s}}$
Absorbed radiation dose	*gray*	$1\,\mathrm{Gy} = 1\ {}^{\mathrm{J}}/_{\mathrm{kg}}$
Plane angle	*radian*	$1\,\mathrm{rad} = 1\ {}^{\mathrm{m}}/_{\mathrm{m}}$
Solid angle	*steradian*	$1\,\mathrm{sr} = 1\ {}^{\mathrm{m^2}}/_{\mathrm{m^2}}$
Luminous flux	*lumen*	$1\,\mathrm{lm} = 1\,\mathrm{cd\,sr}$
Luminance, illuminance	*lux*	$1\,\mathrm{lx} = 1\ {}^{\mathrm{cd}}/_{\mathrm{m^2}}$

For talking about inputs and results, also knowing symbols and names of *SI* prefixes is beneficial. 36 orders of magnitude are covered below: [100]

[100] *SI* prefixes are case sensitive – note M and P below.

Fun fact: Great distances are measured in *kilometer*, then in thousands, millions, or billions of km – Mm etc. are simply not applied. For astronomic distances, *astronomic units* and *light years* are used – some 10^{11} *light years* suffice for everything we know.

DRAFT

Prefix letter	Name	Factor	Exemplary units used
h	*hecto-*	10^{2}	hl, ha, hPa
k	*kilo-*	10^{3}	many
M	*mega-*	10^{6}	many
G	*giga-*	10^{9}	many
T	*tera-*	10^{12}	Tb, TW, TWh
P	*peta-*	10^{15}	PFlops
E	*exa-*	10^{18}	

Prefix letter	Name	Factor	Exemplary units used
d	*deci-*	10^{-1}	dm, dl, dtex
c	*centi-*	10^{-2}	cm, cl
m	*milli-*	10^{-3}	many
µ	*micro-*	10^{-6}	many
n	*nano-*	10^{-9}	many
p	*pico-*	10^{-12}	ps, pF
f	*femto-*	10^{-15}	fs, fm
a	*atto-*	10^{-18}	as

All the conversions featured in U→ are explained in alphabetical order below.[101] There, numeric values are either exact (printed on white) or

---

Prefixes beyond the ones listed here are defined though hardly used. Using exponents of ten is easier as long as no engineers are around.

TN: People knowing ancient Greek and Latin enjoy advantages here.

[101] Within U→, all conversions are displayed as pairs with the *SI* units on the left side. Note that most of these conversions begin or end with *SI* base units, mainly m and kg, or directly derived units like Pa or m^3. This makes further calculations and subsequent conversions most easy.

ATTENTION: Conversions of so-called 'kitchen units' deviate on request of some *British Imperial* natives (applies also to *acres*, *carat, miles*, and *points*): There you must apply the necessary powers of ten before continuing converting. We separated those conversions from the coherent ones in every branch.

The *British Imperial* units in this table cover the most popular in the *UK* and its ex-colonies and territories. The *US* volume units are the customary ones for liquids (they differ for dry volumes for historical reasons). For *SI*, on the other hand, a volume of one *cubic meter* stays one *cubic meter*, be it evacuated or filled with gas, water, mercury, or solid iron. Just *litre* usually measures liquids or gases only. (While we are at it, nobody talks of *kiloliter* like *Merriam-Webster* does instead of *cubic meter*; be aware there are more unusual 'metric' quantities in their 'metric system' table of 2024 – seems they are just too far away still.)

The Chinese units are a selection taken from the market standard *Shìzhì* (市制) valid for mainland China. Note that 10 000 *fēn* = 1000 *cùn* = 100 *chǐ* = 10 *zhàng* = 1 *yǐn*, i.e. a reasonable decimal system. Also note units carrying the same names but used in Hongkong and Macao may differ significantly, as well as those used in other states in

DRAFT

rounded to six significant digits (for your orientation only – your *WP43* uses more precise values wherever applicable). Commas are used as radix marks for better visibility.

Conversion	Calculation	Remarks	Branch
°C → °F	× 1,8 + 32	ϑ [°C] = (T [K] − 273,15) and τ [°F] = (t [°R] − 491,67)	
°F → °C	− 32) / 1,8		
acre → ha	× 4,046 86	This *acre* is based on the '*international foot*' (see below); 640 *acres* = 1 *square mile*	A:
acre → m²	× 40 468,6		
acre$_{US}$ → ha	× 4,046 87	This *acre* is based on the *US survey foot* (see below)	
acre$_{US}$ → m²	× 40 468,7		
atm → Pa	× 101 325	*Atmosphere*	F&p:
au → m	× 1,495 98×10 [11]	*Astronomic unit*	x:
barrel → m³	× 0,158 987	*(US) barrel of oil*, abbr. bbl	V:
bar → Pa	× 100 000	1 mbar = 1 hPa	F&p:
Btu → J	× 1 055,06	*British thermal unit*	E:
cal → J	× 4,186 8	*Calory*	
carat → g	/ 5		m:
carat → kg	/ 5 000		
chǐ → m	/ 3	*'Chin. foot'*; 1 市尺 := 10 *cùn*	x:
cùn → m	/ 30	*'Chin. inch'*; 1 市寸 := 10 *fēn*	
cup$_I$ → mℓ	× 250	1 *intl. (metric) cup* := 250 ml	V:
cup$_{UK}$ → mℓ	× 284,131	1 *cup$_{UK}$* := 10 *fluid ounces (UK)*	
cup$_{US}$ → mℓ	× 236,588	1 *cup$_{US}$* := 8 *fluid ounces (US)*	

this region (e.g., Malaysia, Singapore, Taiwan, or Vietnam) with a major Chinese population. Tibet used to have its own weights and measures before 1959 – then it was not inhabited by a significant number of Chinese either.

Myanmar has got its own units not covered here.

DRAFT

Conversion	Calculation	Remarks	Branch
cwt → kg	× 50,802 4	1 *hundredweight (UK)* := 112 lbs	m:
day → s	× 86 400		Time:
dB → field ratio	$10^{R_{dB}/20}$	*Decibel*	▼
dB → power ratio	$10^{R_{dB}/10}$		
fēn → m	/ 300	1 市分 := 0,1 *cùn*	x:
field ratio → dB	$20\ lg(a_1/a_2)$	Also known as *amplitude*, *voltage* or *current ratio*. a_2 = reference	▼
$floz_{UK}$ → m^3	× 28,413 1×10^{-6}	1 *fluid ounce* := $^1/_{16}$ *pint*	V:
$floz_{UK}$ → mℓ	× 28,413 1		
$floz_{US}$ → m^3	× 29,573 5×10^{-6}		
$floz_{US}$ → mℓ	× 29,573 5		
fathm → m	× 1,828 8	1 *fathom* := 2 *yards*	x:
foot → m	× 0,304 8	So-called '*international feet*' (of 1959); 1 *foot* := 12 *inches*	
gal_{UK} → ℓ	× 4,546 09	*Gallon*; 1 *Imperial gallon* := 8 *Imp. pints* 1 *US liquid gallon* := 231 *cubic inches*	V:
gal_{UK} → m^3	× 4,546 09×10^{-3}		
gal_{US} → ℓ	× 3,785 42		
gal_{US} → m^3	× 3,785 42×10^{-3}		
g → carat	× 5	*Gramm*; 1 g = $\frac{1}{1000}$ kg	m:
g → oz.	/ 28,349 5		
g → troy oz.	/ 31,103 5		
ha → m^2	× 10 000	1 a [*are*] := 100 m^2, 1 ha [*hectare*] := 100 a, 1 km^2 [*square kilometer*] = 100 ha = 10^6 m^2	A:
ha → acre	/ 4,04686		
ha → $acre_{US}$	/ 4,04687		

DRAFT

Conversion	Calculation	Remarks	Branch
hp_E → W	× 746	*Electric horsepower*	P:
hp_M → W	× 735,499	So-called *metric horsepower* [102]	
hp_{UK} → W	× 745,700	*British Imperial horsepower*	
inch → m	× 0,025 4	1 *inch* := 1000 *mil*; but watch the angular *mil!*	x:
inch → mm	× 25,4		
in.Hg → Pa	× 3 386,39	*Inch of mercury*	F&p:
jīn → kg	× 0,5	*'Chin. pound'*; 1 市斤 := 10 liǎng	m:
J → Btu	/ 1 055,06	*Joule*	E:
J → cal	/ 4,186 8		
J → Wh	/ 3 600		
kg → carat	× 5 000	*Kilogram*	m:
kg → cwt	/ 50,802 4		
kg → jīn	/ 0,5		
kg → lb.	/ 0,453 592		
kg → liǎng	/ 0,05		
kg → oz.	/ 0,028 349 5		
kg → short cwt	/ 45,359 2		
kg → short ton	/ 907,185	1 t [*(metric) ton*] := 1000 kg	
kg → stone	/ 6,350 29		
kg → ton	/ 1 016,05	← this is not a metric *ton* !	
kg → troy oz.	/ 0,031 103 5		
km → mile	/ 1,609 344	*Kilometer*	x:
km → nmi	/ 1,852		
lbf → N	× 4,448 22	*Pound force*	F&p:
lbf·ft → Nm	× 1,355 82	Torque: *Pound force times foot*	▼

[102] *TN*: This hp equals ch in French and PS in German, for example.

DRAFT

Conversion	Calculation	Remarks	Branch
lb. → kg	× 0,453 592	*Pound;* 1 lb := 16 *ounces*	m:
liǎng → kg	× 0,05	*'Chin. ounce'* 1 市两 := 0,1 jīn	
lǐ → m	× 500	*'Chin. mile'* 1 市里 := 15 yǐn	x:
l.y. → m	× 9,460 73×10 15	*Light year*[103]	
ℓ → gal_{UK}	/ 4,546 09	1 *litre* := 1 dm^3 = 10^{-3} m^3.	V:
ℓ → gal_{US}	/ 3,785 42	*Litre* (and *millilitre*) are typically used to measure liquids, gases, or empty space. For everything else, use m^3 and its fractions.	
ℓ → quart	/ 0,946 353		
m^2 → acre	/ 40 468,6	*Square meter*	A:
m^2 → $acre_{US}$	/ 40 468,7		
m^2 → ha	/ 10 000		
m^2 → mǔ	× 0,001 5		
m^3 → barrel	/ 0,158 987	*Cubic meter.*	V:
m^3 → $fluid\ oz_{UK}$	/ 28,413 1×10 −6	Feel free to use *litre* for smaller volumes of liquids, gases, or empty space (cf. above).	
m^3 → $fluid\ oz_{US}$	/ 29,573 5×10 −6		
m^3 → gal_{UK}	/ 4,546 09×10 −3		
m^3 → gal_{US}	/ 3,785 42×10 −3		
m^3 → quart	/ 1,136 52×10 −3		
mile → km	× 1,609 344	1 *(statute) mile* := 1760 *yards*	x:
mile → m	× 1 609,344		
mℓ → cup_I	/ 250	*Millilitre*; 1 mℓ = 1 cm^3	V:
mℓ → cup_{UK}	/ 284,131		
mℓ → cup_{US}	/ 236,588		
mℓ → $floz_{UK}$	/ 28,413 1		

[103] Note this is defined by the *IAU* as the distance light travels within one *Julian* year, i.e. 365,25 days.

DRAFT

Conversion	Calculation	Remarks	Branch
mℓ → floz$_{US}$	/ 29,573 5	*Millilitre*	V:
mℓ → pint$_{UK}$	/ 568,261		
mℓ → pint$_{US}$	/ 473,176		
mℓ → table spoon$_{I}$	/ 15		
mℓ → tbsp$_{UK}$	/ 14,207		
mℓ → Tbsp$_{US}$	/ 14,787		
mℓ → tea spoon$_{I}$	/ 5		
mℓ → tsp$_{UK}$	/ 3,552		
mℓ → tsp$_{US}$	/ 4,929		
mmHg → Pa	× 133,322	See Pa.	F&p:
mm → inch	/ 25,4	*Millimeter*	x:
mm → point	/ 0,352 778		
mǔ → m^2	/ 0,001 5	平方亩 (píng fāng mǔ), used for pieces of land in particular.	A:
m → au	/ 1,495 98×10^{11}	*Meter*	x:
m → chǐ	× 3		
m → cùn	× 30		
m → fathm	/ 1,828 8		
m → fēn	× 300		
m → foot	/ 0,304 8		
m → inch	/ 0,025 4		
m → lǐ	/ 500		
m → l.y.	/ 9,460 73×10^{15}		
m → mile	/ 1 609,344		
m → nmi	/ 1 852		
m → pc	/ 3,085 68×10^{16}		

DRAFT

Conversion	Calculation	Remarks	Branch
m → point	/ 3,527 78×10^{-4}	*Meter*	x:
m → survey foot$_{US}$	/ 0,304 801		
m → yard	/ 0,914 4		
m → yǐn	× 0,03		
m → zhàng	× 0,3		
nmi → km	× 1,852	*Nautical mile*	
nmi → m	× 1 852		
Nm → lbf·ft	/ 1,355 82	*Newton meter* (for torques)	▼
N → lbf	/ 4,448 22	*Newton*	F&p:
oz. → g	× 28,349 5	*Ounce*	m:
oz. → kg	× 0,028 349 5		
Pa → atm	/ 101 325	*Pascal*; 1 hPa = 1 mbar For all real-world applications (i.e. within experimental errors), *torr* are equivalent to *millimeter of Hg*. Possible differences are merely calculatory. Please see also the *NIST* document linked on p. 160.	F&p:
Pa → bar	/ 100 000		
Pa → in.Hg	/ 3 386,39		
Pa → mmHg	/ 133,322		
Pa → psi	/ 6 894,76		
Pa → torr	/ 133,322		
pc → m	× 3,085 68×10^{16}	*Parsec*	x:
pint$_{UK}$ → mℓ	× 568,261	1 *Imp. pint* := $^1/_8$ *gallon (UK)*	V:
pint$_{US}$ → mℓ	× 473,176	1 *US liq. pint* := $^1/_8$ *liq. gallon*	
point → m	× 3,527 78×10^{-4}	1 *(typographical) point* := $^1/_{72}$ *inch*	x:
point → mm	× 0,352 778		
power ratio → dB	$10\ lg(^{p_1}/_{p_2})$	p_2 = reference power	▼
psi → Pa	× 6 894,76	*Pound per square inch*	F&p:

DRAFT

Conversion	Calculation	Remarks	Branch
quart → ℓ	× 0,946 353	1 *US liq. quart* := $^1/_4$ *liq. gallon*	V:
quart → m^3	× 9,463 53×10 $^{-4}$		
short cwt → kg	× 45,359 2	1 *short hundredweight (US)* := 100 lbs	m:
short ton → kg	× 907,185	1 *short ton (US)* := 2000 lbs	
stone → kg	× 6,350 29		
survey $foot_{US}$ → m	× 0,304 801	1 *US survey foot* := $\frac{1200}{3937}$ m	x:
s → day	/ 86 400		Time:
s → year	/ 31 556 952		
table $spoon_I$ → mℓ	× 15	1 *internatl. (metric) tablespoon*	V:
$tbsp_{UK}$ → mℓ	× 14,207	1 *tablespoon* := $^1/_2$ *fluid ounce*	
$Tbsp_{US}$ → mℓ	× 14,786 8		
tea $spoon_I$ → mℓ	× 5	1 *internatl. (metric) teaspoon*	
tsp_{UK} → mℓ	× 3,552	1 *British teaspoon* := $^1/_4$ *tablespoon (UK)*	
tsp_{US} → mℓ	× 4,928 92	1 *US teaspoon* := $^1/_3$ *tablespoon (US)*	
ton → kg	× 1 016,05	1 *Imperial ton* := 200 cwt	m:
torr → Pa	× 133,322	See Pa.	F&p:
troy oz. → g	× 31,103 5	*Troy ounce*	m:
troy oz. → kg	× 0,031 103 5		
Wh → J	× 3 600	*Watt-hour*	E:
W → hp_E	/ 746	*Watt*	P:
W → hp_M	/ 735,499		
W → hp_{UK}	/ 745,700		

DRAFT

Conversion	Calculation	Remarks	Branch
yard → m	× 0,914 4	1 *yard* := 3 *feet*	**x:**
yǐn → m	/ 0,03	1 市引 := 10 *zhàng*	
year → s	× 31 556 952	= 365,242 5 × 24 × 60^2	**Time:**
zhàng → m	/ 0,3	1 市丈 := 10 *chǐ*	**x:**

Some units more, being really simply convertible, follow:

- $1\ ångström \equiv 1\ \text{Å} = 10^{-10}\ \text{m} = 0{,}1\ \text{nm}$ may be convenient for wavelengths and atomic diameters (howsoever the latter are specified)
- $1\ fermi \equiv 1\ fm = 10^{-15}\ m$; although a regular part of a *meter*, this is found as a separate unit on some calculators
- $1\ barn \equiv 1\ \text{b} = 10^{-28}\ \text{m}^2$ for cross sections in nuclear and particle physics (note the name; *millibarn* and *nanobarn* are 'common' units in reactions there)
- $1\ tex = 10^{-6}\ {}^{\text{kg}}/_{\text{m}}$ for yarn density in textile industry (where *decitex* is common for nylons), and
- $1\ muggeseggele \approx 3 \ldots 1\ \mu\text{m}$ for Swabian precision mechanics.[104]

[104] See https://en.wikipedia.org/wiki/Muggeseggele for more about this (frankly, I did not expect an article about this pretty local unit there at all).

Side note: I enjoyed the privilege of experiencing the *muggeseggele* in its natural habitat at the premises of a Swabian manufacturer of precision components for the textile industry – founded in 1852, family-owned until today, and one of several 'hidden champions' (i.e. world market leaders) located in south-western Germany (in a hilly region where people rightfully claim *wir können alles außer hochdeutsch* (translated: *we can do/make everything but speaking standard German*)) – and its affiliates worldwide for many years.

It was definitely fun listening to, e.g., Portuguese or Indian staff talking about *muggeseggele* – you can't translate it. On the shop floor, 1 *muggeseggele* meant significantly less than 1 *hundertstel* (= 0.01 mm) – but it could still be reached manually by a well-trained technician tightening the proper bolt using appropriate manual (!) torque. It was a real difference, although any quantity deviating from just one *muggeseggele* was never observed in the wild by me (may be caused by language: German *ein Muggeseggele* translates to *a muggeseggele* as well as to *one muggeseggele*).

DRAFT

The 22 angular conversions provided are (cf. *OMS 2*):

…	Remarks		
DEG→	Takes an *integer* or *real number* x as an angular input in *decimal* or *sexagesimal degrees*, resp., and converts it to the current *ADM.*		
D.MS→			
D.MS→D	Takes an *integer* or *real number* x as an angular input in *sexagesimal degrees* (formatted `ddddd.mmsshh`) and converts it to an *angle* in *decimal degrees* (corresponding to vintage H.MS→H).		
D→D.MS	Takes an *integer* or *real number* x as an angular input in *decimal degrees* and converts it to …	… *sexagesimal degrees* (corresponding to vintage H→H.MS).	
D→MIL		… *mils.*	
D→R		… *radians.* [105]	
GRAD→	Takes an *integer* or *real number* x as an angular input in …	… *grades/gon* …	… and converts it to the current *ADM.*
MIL→		… *mils* …	
MULπ→		… *multiples of* π …	
RAD→		… *radians* [105] …	

In series production, quality microscopes and projectors featuring magnifications times 20 to 50 were typical tools for machine setup control. By the way, the staff there never talked of high quality or precision – machines, instruments, tools, and products were 'usable' or 'just right' at best. Compare to other (e.g., marketing driven) environments and the outcomes there, in particular in medium to long range.

Specifying the *muggeseggele* any further is certainly a challenge for a dedicated metrology engineer with very long wind.

Find more exotic units in a guide of *NIST* (https://physics.nist.gov/cuu/pdf/sp811.pdf). In its appendix B, it covers on 30 full pages not only many outdated units like *footcandles*, *oersted*, and *tons of refrigeration* but also tells comprehensively how to deal with them. Numerous *Imperial* units, however, are missing therein.

[105] Note that large numeric inputs in trigonometric functions will be reduced before calculating (cf. the *IOI* and *OMS 2*). Such reductions in *radians* can't be exact since *radians* can neither represent full circles nor simple fractions of them like π, $\pi/2$, $\pi/3$, etc. exactly but with an accuracy of 34 digits 'only'.

If you want to avoid rounding errors caused by that but must keep π in play, *multiples of* π may be a better choice for the *ADM*. The price to pay is a conversion of

DRAFT

...	Remarks
MIL→D MIL→R	Takes an *integer* or *real number* x as an angular input in *mils* and converts it to *decimal degrees* or *radians*, respectively. [105]
Mπ→R	Takes an *integer* or *real number* x as an angular input in *multiples of* π and converts it to *radians*. [105]
R→D R→MIL R→Mπ	Takes an *integer* or *real number* x as an angular input in *radians* and converts it to *decimal degrees*, *mils*, or *multiples of* π, respectively. [105]
→DEG →D.MS →GRAD →MIL →MULπ →RAD	Takes an *integer* (*DT 1*) or *real number* (*DT 2*) x as an angular input in the current *ADM* and converts it to *decimal degrees*, *sexagesimal degrees*, *grades/gon*, *mils*, *multiples of* π, or *radians*, respectively. [105] If x is a tagged *angle* (*DT 4*), then tag information is used in conversion (e.g., if $x = 1.5\pi$ then →GRAD will return 300^g regardless of current *ADM*). If x is neither of *DT 1*, *2*, nor *4*, error 24 will be thrown.

Angular output is tagged always.

In some industries, also angular units may carry prefixes as listed on p. 152: some military instruments may indicate *milliradians*, for instance.

the reduced angle into *radians* just before the final computation of the *Taylor* series. See *Background Facts and Considerations*, Angles, on p. B-6.

[106] Solution of f. 78: In this very simple model, an average volume of $150\ (l.y.)^3$ is available for each star in our home galaxy, i.e. a sphere with a diameter of 6 *l.y.* If that model would cover all important properties of a galaxy, nothing spectacular is expected to happen in a galactic collision.

For estimating the consequences of such a collision more realistically, take the orientation of both galaxies relative to each other into account when they collide. And star density is not uniform but highest in the galactic centers and decreases from there outwards. Furthermore, there use to be giant amounts of dust and free gas (although very thin) between the stars.

Photographs of colliding galaxies are available in the net.

DRAFT

SECTION 3: CALLING AND EXECUTING OPERATIONS

As mentioned above, the number of *items* featured on your *WP43* is far too large to fit them on the keyboard. Hence, there are several ways to call such an *item*.

You already know how to call an *item* appearing on the keyboard or in a *menu* or *catalog*. In *OMS 6*, you have learned about storing *items* in user *menus* and / or assigning them to specific locations on your *WP43*. There is one more way you may call an operation, described in next chapter.

In the two chapters following thereafter, we will list all the functions requiring parameters and those changing *data types*.

Using XEQ for Executing Operations

Instead of picking an operation from a *menu* or *catalog*, you may also call it by specifying its *name*:

1. Press [XEQ] [α]. You enter *AIM* this way (cf. p. 165 for the *virtual keyboard* applicable).
2. Type the *name* of the function wanted. Case is important, subscript or superscript is not. Press [ENTER↑] if the *name* is shorter than 7 *characters*.
3. Your input will be examined; if the specified operation exists, …
   a. then you will be prompted for trailing parameters, if applicable (see next chapter); then the function will be executed.
   b. else `No such function` will be thrown.

This may be neither the easiest nor the fastest way to access operations (correct spelling is required) but it is allowed for sake of tradition.

DRAFT

Command Parameter Input and Closing It

What will happen when a particular key is pressed while command parameter input is open (see pp. 166ff for input in **X** instead)? Note that a *character* may be a letter, digit, punctuation mark, symbol, etc. The table lists the respective keys beginning top left on the keyboard:

Keystrokes	Situation	Meaning
A, B, C, D, I, J, K, L, T, X, Y, Z	addressing	Enters the address of a *global GP register* or *user flag*.
A … Z 🟦A … 🟦O 🟨0 … 🟨9	entering a *label*, *system flag*, *variable,* string constant, or comment with *AIM* open	Appends the corresponding Latin or Greek letter or digit to the *item* pending. Use ▼ and ▲ to switch case for letters. Use ← and → to move the cursor. See the *virtual keyboard* on p. 165; see also p. 166 for limitations applying to comments and string constants.
ENTER↑	arbitrary parameter input pending	If there is no input yet, assumes the default, if applicable. Else closes pending input, interprets it as expected for this command, and executes it (cf. *OMS 1*).
⬅	arbitrary parameter input pending	Deletes the last *character* entered. If nothing is left, cancels the pending command, returning to the state of your *WP43* as it was before that command was called.
0 … 9	addressing or specifying	Enters a numeric parameter, an address, or a *local label*. See *OMS 1* and *3* for valid number ranges.
.	addressing	Header for *local registers* or *local user flags*.
EXIT	arbitrary parameter input pending	If there is an open *menu*, closes it. Else cancels pending command input, returning to the status of your *WP43* as it was before the current command was called.

DRAFT

Virtual keyboard in *alpha input mode* (*AIM*). *AIM* is also active while …

- editing equations (see pp. 175f) as well as
- while a *catalog* is open, so you can use all accessible letters for alphabetic searching (cf. pp. 136f).

Note there is an ‘alpha helper’ on the backplate of your *WP43* (see p. B-31).

Alphanumeric Input in X and Closing It

The following table shows what will happen when a particular key is pressed with (alpha-) numeric input being open (turn to p. 164 for command parameter input instead). It lists the respective keys on the keyboard top left to bottom right:

Keystrokes	in mode(s)	Meaning
A … Z blue A … blue O	A, α	Appends the corresponding Latin or Greek letter to the *text string x*. Use ▼ and ▲ to switch case. Cf. previous page and *OMS 2*. Note that input strings in *RUM* may be 2 rows long, string constants in *PEM* almost 4 rows or 196 *characters* (whatever is reached first), and REM-comments up to 140 *characters*.
#	A, α	Appends # to the *text string x*.
# *base*	¬ (A, α)	Closes input of a *short integer* in X.[107]
d.ms		sexagesimal *angle* in X.[107]
.d		*date* in X.[107]
h.ms		sexagesimal *time* in X.[107]
blue TRI (( )	A, α	Appends (or) to the *text string x* or inserts it into the current equation.
blue ln ( ))		
blue e^x	A, α	Appends e^ to the *text string x* or inserts e^( ) into the current equation.

[107] Sexagesimal *angles* shall be entered like `d.mmssff` (> 1 digit is allowed for *degrees*).

Input of *times* shall be like `h.mmssf` (*hours* and fractions of *seconds* may both comprise > 1 digit).

See *OMS 2* for examples and *App. B* for numeric limitations.

At closure, input will be checked – *seconds* (or *minutes*) > 60 will be carried to *minutes* (or *hours*). Illegal digits (e.g., **8** in octal input or **C** in decimal), bases, numbers, or *characters* found, or out-of-range conditions detected will cause an error thrown then (see also the description of ENTER↑ on next page and the error messages compiled in *App. C*).

DRAFT

Keystrokes	in mode(s)	Meaning
[f] [√x]	A, α	Appends a root symbol √ to the *text string* **x** or inserts √(▪) into the current equation.
[f] [H] (CH)	A, α	Appends CH or ch to the *text string* **x**.
[f] [R↓] (⇩)	A, α	Makes the next *character* a subscript, if applicable.
[CC]	¬ (A, α)	Closes input of the first (i.e. *real* or *magnitude*) part of a *complex number* in **X** and waits for input of its second part (i.e. imaginary or *phase*, see *OMS 2* and *App. 1*). Any additional [CC] in input will be ignored.
[|x|]	A, α	Appends one bar | to the *text string* **x** or inserts | ▪ | (for ABS) into the current equation.
[ENTER↑]	arbitrary input pending	If input was expected but not entered, cancels entry. Else closes input in **X** and checks the following conditions top-down: 1. If **x** is alphanumeric (i.e. **x** contains at least one non-numeric *character* except [.]), takes it as a *text string*. 2. Else (i.e. **x** is purely numeric) if **x** contains one [CC], takes it as a *complex number*. 3. Else if **x** contains two [.], takes it as a *fraction*. 4. Else if **x** contains a [.] or an [E], takes it as a *real number*. 5. Else (i.e. **x** contains neither [CC] nor [.] nor [E]), tests **x** for [#]: a. If it contains a [#] and a valid base trailing it then takes **x** as a *short integer* of said base; b. else looks up if the previous entry was a *short integer*: if true then takes the new input as another *short integer* of the same base; else takes the new input as a *long integer*. Then checks the new input (according to the condition met) as outlined in f. 107 on previous page and interprets it. Finally, unless an error had to be thrown, copies **x** into **Y**.

DRAFT

Keystrokes	in mode(s)	Meaning
[f] [x≷y]	A, α	Appends ≷ to the *text string* ***x***.
[+/−]	¬ (A, α)	Changes the sign of the number entered. Affects the exponent if pressed after [E].
[f] [+/−]	A, α	Appends ± to the *text string* ***x***.
[E]	¬ (A, α)	Closes input of the mantissa and waits for input of the exponent (see *OMS 1*). Any additional [E] in input will be ignored.
[f] [E] (↑)	A, α	Makes the next *character* a superscript, if applicable.
[←]	arbitrary input pending	Deletes the *character* left of the cursor. If nothing is left, cancels pending input, returning to the status of your *WP43* as it was before that input was started.
[f] [/]	A, α	Appends / to the *text string* ***x*** or inserts this symbol into the current equation.
[XEQ] (:)	A, α	Appends a : to the *text string* ***x***.
[f] [XEQ] (QU)	A, α	Appends QU or qu to the *text string* ***x***.
[f] [×]	A, α	If MULT× is set then appends ×, else · to the *string* ***x***. Editing an equation, inserts a · always.
[f] [−], [f] [+]	A, α	Appends the respective sign to the *text string* ***x*** or inserts this symbol into the current equation.
[INTS] A … F [BIT] [▲] A … F	¬(A, α)	Numeric input for *integer* bases >10, appending the corresponding digit to ***x***. See *OMS 2*. Digits will be checked when ***x*** is closed (cf. the description of [ENTER↑] on previous page).
[0] … [9]	¬ (A, α)	Standard numeric input, appending the corresponding digit to ***x***. Note you can enter … • up to 34 digits plus a sign in the mantissa[108] and up to four digits plus a sign in the exponent for a *real number* or any part of a *complex number*, • an arbitrary number of digits plus a sign for a *long integer*,

[108] You can enter even more digits but it will make no difference.

DRAFT

Keystrokes	in mode(s)	Meaning
		• up to 64 bits for a *short integer*, or • up to 16 digits for the nominator and up to 4 digits for the denominator of a fraction.
0 (?)	A, α	Appends **?** to the *text string* ***x***.
f 0 … f 9	A, α	Appends the respective digit to the *text string* ***x*** or inserts it into the current equation.
f 0 (PRINT)	A, α	Appends 🖷 to the *text string* ***x***.
▲	α	Turns to upper case for the letters following.
▼	A	Turns to lower case for the letters following.
.	¬ (A, α)	For D.MS input, separates *degrees* from *minutes*, *seconds*, and *hundredths of seconds*, so input format is `dddd.mmsshh` d.ms (cf. p. 166 and *OMS 2*). For *time* input, separates *hours* from *minutes*, *seconds*, and fractions of *seconds*, so input format is `hhhh.mmssffff` h.ms (cf. p. 166 and *OMS 2*). Else inserts a radix mark as selected.
Second .	¬ (A, α)	A 2nd . in input signals a fraction. See *OMS 2* for examples. The 2nd . separates nominator and denominator in input. Any further . in input will be ignored. Note you can't enter E after you entered . twice but you may delete the 2nd dot while editing the input.
.	A, α	Appends a **,** to the *text string* ***x***.
f .	A, α	Appends a **.** to the *text string* ***x***.
R/S	🏃, program waiting for input	Closes pending input and starts checking and interpreting it like ENTER↑ on p. 167. Resumes program execution.
	A, α	Appends a blank space to the *text string* ***x***.
f R/S (!)	A, α	Appends an **!** to the *text string* ***x*** or to the current equation.

DRAFT

Keystrokes	in mode(s)	Meaning
EXIT	arbitrary input pending	If there is an open *menu*, closes it. Else closes pending input and starts checking and interpreting it like ENTER↑ on p. 167.

There are many more *characters* you can enter via the three alpha *menus* or via CATALOG CHARS (cf. pp. 134 and 147ff). Call FBR for browsing the entire *character* sets provided.

Any command not specified in the table above will close pending input and release it for interpretation like ENTER↑, R/S, and EXIT do.

Operations Requiring Trailing Parameters

Traditionally, *RPN* operations fetched their parameters from the *stack*, specifying what they shall do precisely (cf. *OMS 1*). Many *WP43* functions, however, require at least one following numeric or alphanumeric parameter now. Three tables below list these operations.

Note that for any command CMDX requiring 1 trailing parameter, you can enter

CMDX → X

and it will fetch its parameter from X instead like a good old *RPN* command.

Operations requiring 1 trailing parameter	Numeric parameter	Alphanumeric parameter
AGRAPH BUZZ CONVG? DEC DSE DSL DSZ INC INPUT ISE ISG ISZ KEY? KTYP? PLAY PUTK RCL RCL... STO STO... t↓ VIEW x→α x↓ x=? x≠? x≈? x<? x≤? x≥? x>? y↓ z↓ αLENG? αPOS? αRL αRR αSL α→x	*Register #* [109]	*Variable name*

[109] # stands for *number* in this table, wherever applicable.

Operations requiring 1 trailing parameter	Numeric parameter	Alphanumeric parameter
ALL DSP ENG FIX GAP RDP RSD SCI SDL SDR	Number of decimals	
ASR MASKL MASKR RL RLC RR RRC SL SR WSIZE	Number of bits	
BACK CASE SKIP	# of program steps	
BC? BS? CB FB SB	Bit number	
BestF	Fit model code	
CF FC? FC?C FC?F FC?S FF FS? FS?C FS?F FS?S SF	*User flag* number	*System flag name* or *shortcut*
CNST	Constant number	
DSTACK	# of stack registers	
ERR MSG	Error number	
f'(x) f''(x) GTO LBL LBL? PGMINT PGMSLV VARMNU Π_n Σ_n and also **SOLVE** and $\int$ in *RUM*		*Label*
GTO		*Label*
GTO.	# of the program step	
INDEX MVAR M.DIM M.EDIN also **SOLVE** and $\int$ in *PEM*		*Variable name*
LocR	# of local registers	
PAUSE	Number of ticks	
RM	Mode number	
SIM_EQ	Number of unknowns	
TDISP	Time display precision	
TONE	Tone number	
VOL	Volume	

DRAFT

Operations requiring 1 trailing parameter	Numeric parameter	Alphanumeric parameter
XEQ		*Label* or function name
→INT	Base	

Operations requiring 2 trailing parameters	1st parameter	2nd parameter
ASSIGN	*Item*	Sequence of keystrokes to access the target
KEYG KEYX	Key number (1 ... 18)	Program *label*

Operation requiring 4 trailing parameters	1st to 4th parameter
⇶	Name of *stack register*

Operations Allowing For Indirect Addressing

AGRAPH ALL ASR AVIEW BACK BC? BestF BS? CASE CB CF CNST CONVG? DEC DSE DSL DSP DSTACK DSZ ENG ERR FB FC? FC?C FC?F FC?S FF FIX FS? FS?C FS?F FS?S f'(x) f"(x) GAP GTO INC INDEX INPUT ISE ISG ISZ KEYG KEYX KEY? KTYP? LBL? LocR MASKL MASKR MSG M.DIM M.EDIN PAUSE PGMINT PGMSLV PLAY PROMPT PUTK RCL... RDP RL RLC RM RR RRC RSD SB SCI SDL SDR SF SIM_EQ SKIP SL SOLVE SR STO... TDISP TONE t⇶ VARMNU VIEW VOL WSIZE XEQ x→α x⇶ x=? x≠? x≈? x<? x≤? x≥? x>? y⇶ z⇶ αLENG αPOS αRL αRR αSL αSR α→x Π_n Σ_n and **∫**.

Note that correct indirect addressing can't be verified in *PEM*. This is because the content of the pointer *RoV* at execution time is unknown at the time the respective program step is entered.

DRAFT

Example:

Assume you are entering a program step

STO →POINT

then your *WP43* can't know now what POINT will contain at (later) execution time of this step.

Operations Changing Data Types

Most functions will return data of the same type they operate on. Some, however, will change the *DT* of the contents of the lowest *stack register(s)* regardless of specific input values, as mentioned at various locations in the *OM* (functions printed on yellow below will return the input *DT* wherever possible). Find all these operations below:

Input *DT*	Operations	Output *DT*	Output *registers*
1	1/x AGM ALL cos ENG erf erfc e^x FIX gd gd^{-1} $J_y(x)$ LNβ LNΓ MANT SCI sin tan W_m W_p W^{-1} β(x,y) Γ(x) Γ_{xy} γ_{xy} Δ% →REAL % %MRR %T %Σ %+MG and || as well as all distributions, unit conversions, and orthogonal polynomials	2	**X**
	$\sqrt[3]{x}$ lb lg ln $LOG_x y$ $\sqrt[x]{y}$ $\sqrt{x}$	2 or 3	
	→POL →REC	2	**X, Y**
	SLVC SLVQ	2 or 3	**X, Y, Z**
	f'(x) f''(x) SOLVE	2	**X, Y, Z, T**
	Σ+ Σ-	2	**X, Y**, STATS
	arccos arcsin arctan and all angular conversions	4	**X**
	→H.MS	5	
	J→D →DATE	6	
	x→α	7	

DRAFT

Input *DT*	Operations	Output *DT*	Output *registers*
1	FACTOR	8	**X**
	M.GET M.NEW	8 or 9	
	→INT	10	
2	AND ceil DATE→ DAY D→J EXPT floor IDIV IDIVR IP MONTH NAND NEXTP NOR NOT OR RANI# ROUNDI sign WDAY XNOR XOR YEAR	1	**X**
	DECOMP	1	**X, Y**
	SLVC SLVQ	2 or 3	**X, Y, Z**
	RE→CX	3	**X**
	arccos arcsin arctan and all angular conversions	4	
	→H.MS	5	
	x→DATE →DATE	6	
	x→α	7	
	M.GET M.NEW	8 or 9	
	→INT	10	
3	ABS cross IM RE |x| ∢	2	**X**
	CX→RE	2	**X, Y**
	SLVC SLVQ	2 or 3	**X, Y, Z**
4	cos sin tan sinc sincπ →REAL	2	**X**
5	→HR	2	**X**
6	DAY D→J MONTH WDAY YEAR	1	**X**
	DATE→	1	**X, Y, Z**
	→REAL	2	**X**
7	α→x	1	**X**

DRAFT

Input *DT*	Operations	Output *DT*	Output *registers*
8	M.DIM?	1	**X, Y**
	DET dot ENORM |M|	2	**X**
	Σ+	2	**X, Y,** STATS
	V⊿	4	**X**
9	M.DIM?	1	**X, Y**
	ENORM	2	**X**
	DET dot |M|	3	
	ABS IM RE ROUNDI |x|	8	**X**
10	sign	1	**X**
	e^x LN $LOG_x y$ →REAL	2	
	x→α	7	
	FACTOR	8	

Operations in Equations

$\sqrt[3]{}$() ABS() ACOS() arccos() arcosh() arcsin() arctan() arsinh() artanh() ASIN() ATAN() atan2(:) B_n() B_n*() ceil() COMB(:) cos() cosh() erf() erfc() $e^{\wedge}$() FIB() floor() g_d() g_d^{-1}() H_n() H_{np}() IDIV(:) lb() lg() L_m() $L_{m\alpha}$() ln() lnΓ() log(:) max(:) min(:) MOD(:) PERM(:) P_n() RMD(:) sin() sinc() sincπ() sinh() SQRT() tan() tanh() T_n() U_n() W_m() W_p() W^{-1}() $\sqrt[x]{}$(:) Γ() ζ() + - · / ^ ! $|_{\blacksquare}|$ √()

Equations shall be written in algebraic notation and will be evaluated following *PFEMDAS*. Functions and operations shall be spelled as they are in the *IOI* but followed by ( ), except the following (cf. p. 165):

DRAFT

Items	In equations	Examples in equations[110]
1/x	1 /	1 / b 1 / (17 + y)
10^x 2^x y^x	^	10^7.36 2^10 23.4^(-5) 2^(y · 1.2 + 3)
x^2 x^3	^	x^2 y^3 (3.4 · z)^2
e^x $(-1)^x$	e^, (-1)^	e^x e^(-6.7) e^(y - 2) (-1)^4.7
x!	!	z! 17! 23.4567! (x+3)!
|x| ABS	| |	|y - 7| |z|
$\sqrt{x}$ SQRT	√( )	√(123.4)
$\sqrt[3]{x}$	∛( )	∛(614.125)
$\log_x y$	log(:)	log(625:5)
$\sqrt[x]{y}$	ˣ√(:)	ˣ√(625:4) ˣ√((x + 0.5):6)

Functions with two parameters like COMB shall read like **COMB(y:x)** in equations, with a colon separating both parameters and the 1st value corresponding to ***y***, the 2nd to ***x***.

Square roots, absolute values, and exponentials can be entered quickly using their function keys [yellow] [√x], [|x|], and [yellow] [e^x] inserting √(▪), | ▪ |, and e^ ▪ , respectively (cf. *OMS 4*). All other functions can be either

- typed and spelled as listed above – watch upper and lower case; super- and subscripts can be replaced by standard *characters* in entry: **HnP** will be decoded as H_{nP}, for instance; or
- picked from CATALOG'FCNS.

There are no implicit multiplications. Multiplication dots are used for optimum visual differentiation from the letter **x**.

Constants may be entered numerically (like **1.2** or **345.69E78**, the latter with a capital **E**) or spelling their *names* as they read in CONST (cf. pp. 113ff) or picked from CATALOG'CONST.

ATTENTION 1: The plain lower case **e** recalls the *elementary charge*

[110] Note that **x**, **y**, and **z** are *variables* in these examples.

DRAFT

– this is not *Euler's* **e**! Also **a**, **c**, **F**, **G**, **h**, **k**, **R**, **α**, **γ**, **Φ**, and **ω** are constants and hence must not be taken for *variables* in equations.

ATTENTION 2: Watch proper spelling of functions, constants, and *variables*. Your WP43 will definitely not decode **SIN(α)** as you may have meant it, but as you wrote it. How shall it know any better?

DRAFT

APPENDIX A: HARDWARE

Dimensions: wedge-shaped 77 mm × 144 mm × 13 mm (at top above display) or 8 mm (at bottom).

Mass: 180 g incl. battery.

LCD: monochrome high contrast (14:1) transflective memory display with an active area of 58.8 mm × 35.3 mm, 400 × 240 square pixels, and dot pitch 147 µm.

Processor: low power *ARM Cortex-M4F* (*STMicroelectronics STM32L476*) incl. real-time clock, running at 25 MHz on battery power or at 80 MHz with power supplied through the *USB* port (see *https://www.st.com/content/st_com/en/products/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus/stm32-ultra-low-power-mcus/stm32l4-series/stm32l4x6.html* for more).

Memory: 1 MB *FM* and 128 kB *RAM* (incl. 96 kB user *RAM*) on *CPU*. 8 MB external *QSPI FM* (~ 6 of these 8 are formatted as *FAT* filesystem for loading and saving programs as well as backup of *WP43 states*, accessible via *USB* as *FD*; see *App. F*).

Power supply: 3 V by one *CR2032* lithium coin cell (battery life up to 3 years, non-rechargeable); optionally powered through the *USB* port (a good battery shall be installed); typical average currents drawn: power on and busy: 4.2 mA; idle: 0.1 mA; power off: 3 µA.

Buzzer: piezo-electric with 4 kHz resonance frequency, tunable from 1 Hz up to 20 kHz in steps of 1 Hz.

Connectivity: infrared port compatible with *HP-82240A/B* printers,

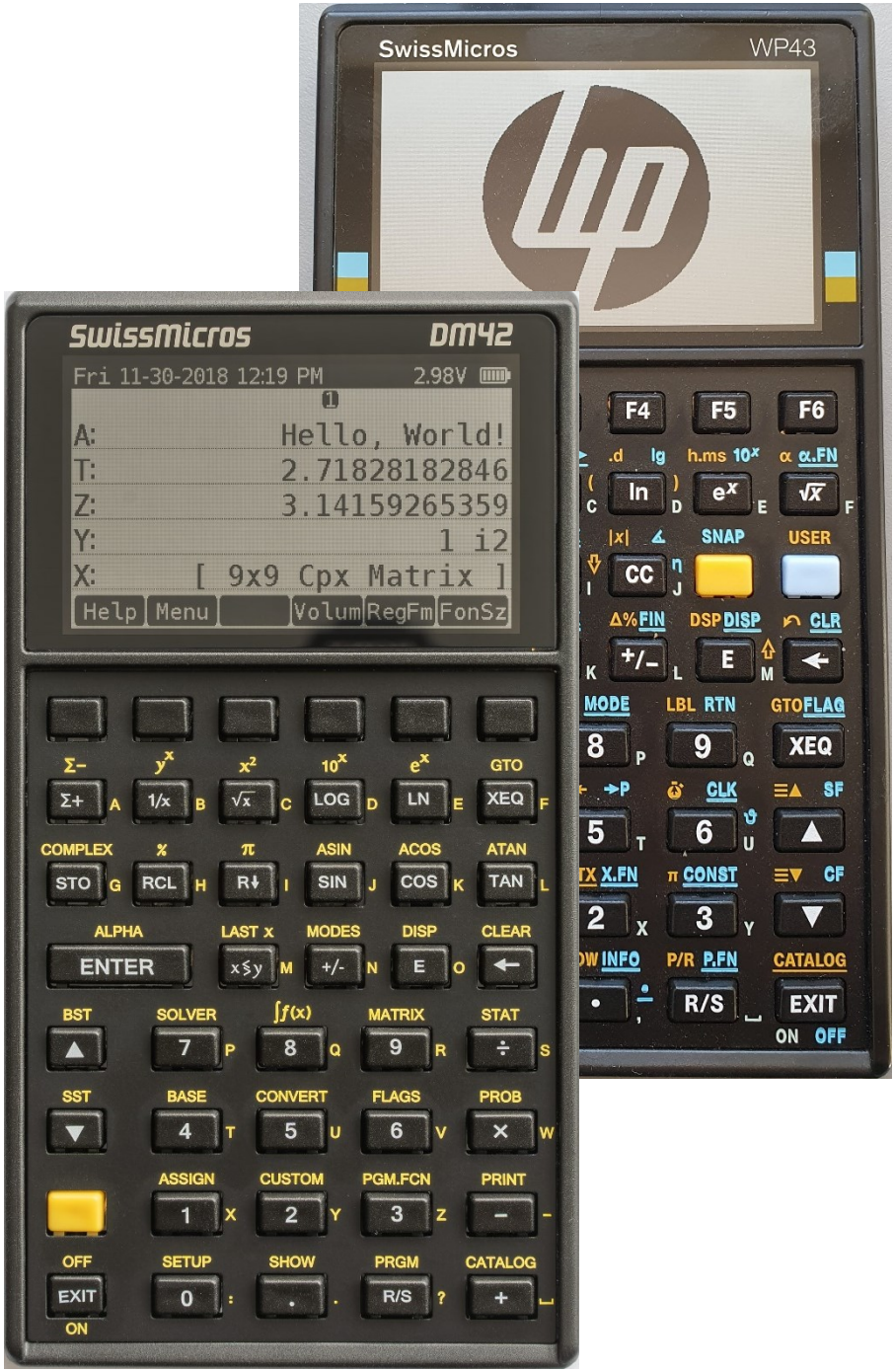
DM42: Micro-USB (-B) port connecting to a host as external mass storage device.

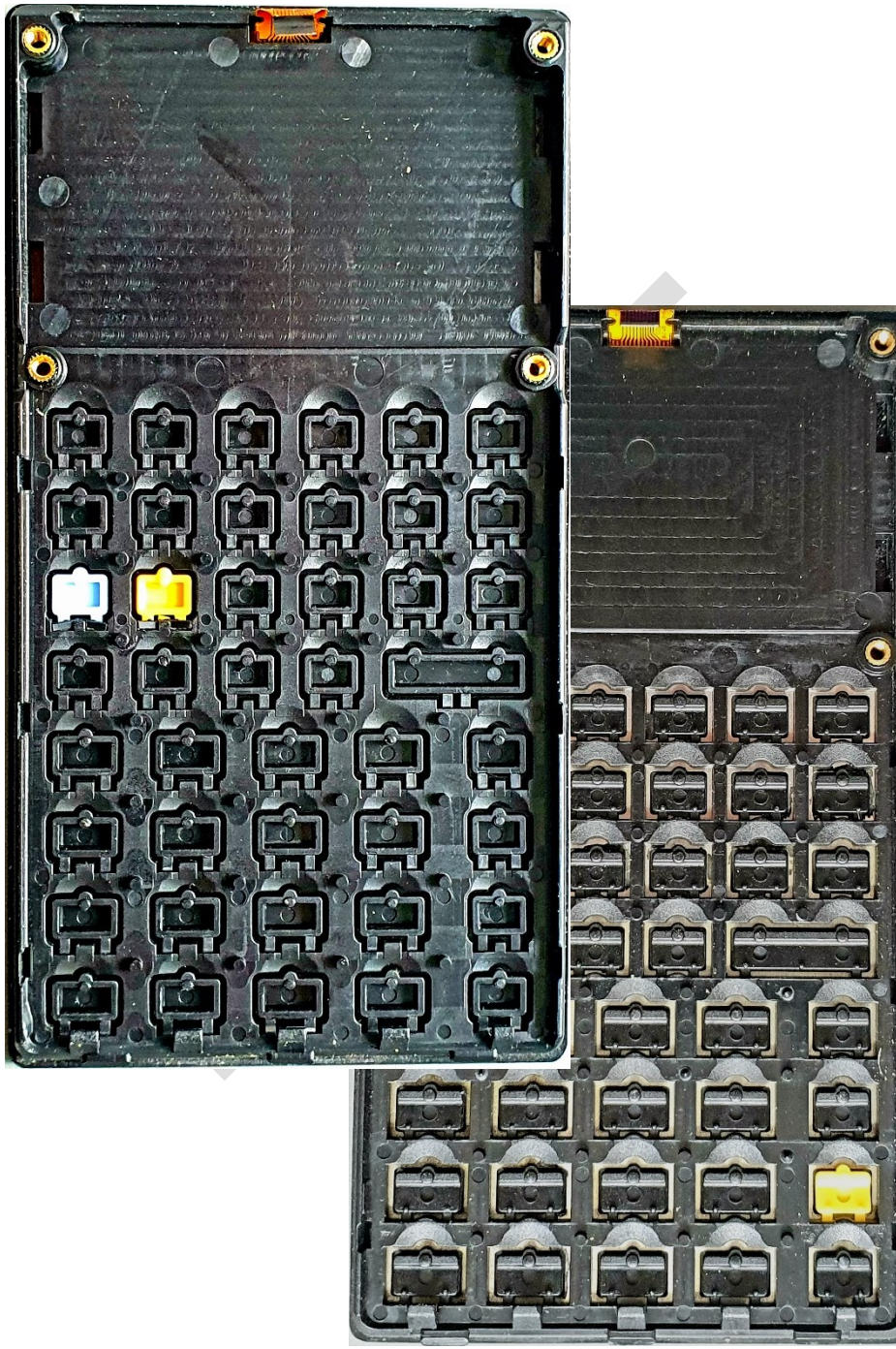
DM42n: USB-C port instead.

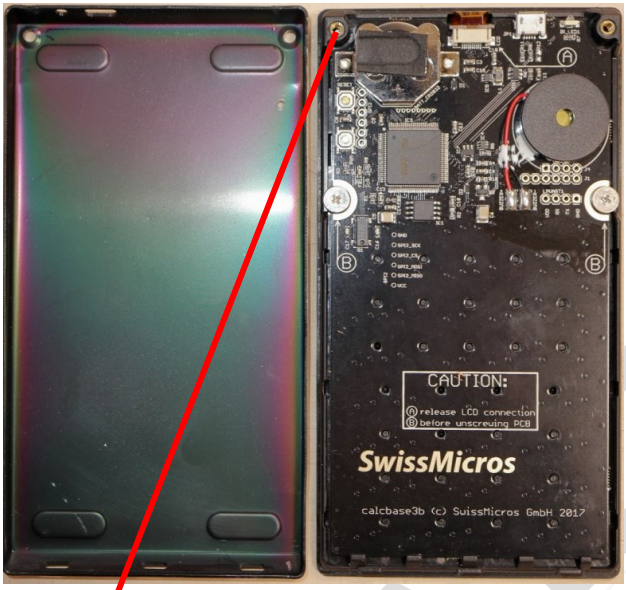
Keyboard overlays: three short slots on either side of the keyboard are provided in the calculator frame for easily fixing your overlay sheets (≤ 0.35 mm thick) with your personal layouts printed on. See p. 184 for a drawing. Look up *OMS 6* for inspiration.

Seven pictures of the hardware are shown on the next four pages. The keyboard views on the next two pages are printed approximately to scale. There you can see why the positions of the keys (in particular the prefixes) are fixed.

DRAFT

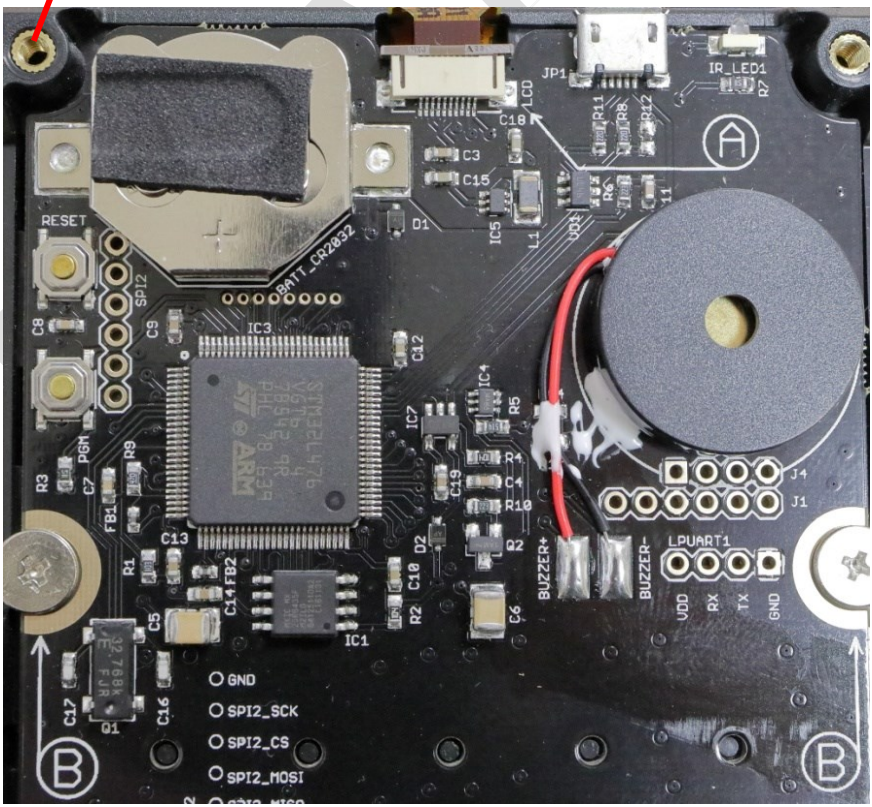

SwissMicros
WP43
SwissMicros
DM42
Fri 11-30-2018 12:19 PM
2.98V
A: Hello, World!
T: 2.71828182846
Z: 3.14159265359
Y: 1 i2
X: [9x9 Cpx Matrix]
Help
Menu
Volum
RegFm
FonSz
ENTER
EXIT
ON
OFF





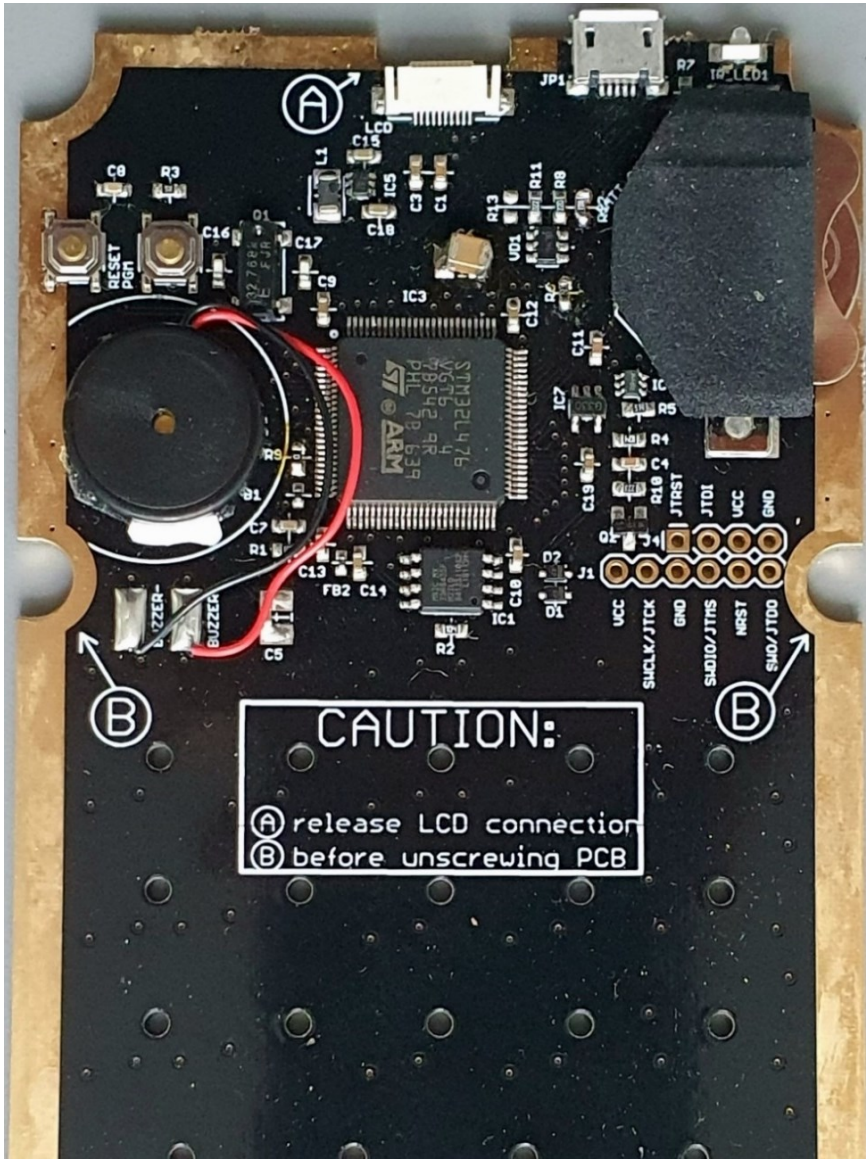
See here the internals of a *DM42* of 2022. Unfasten the two countersunk M2,5×5mm bolts at the top of the calculator backside to get there.

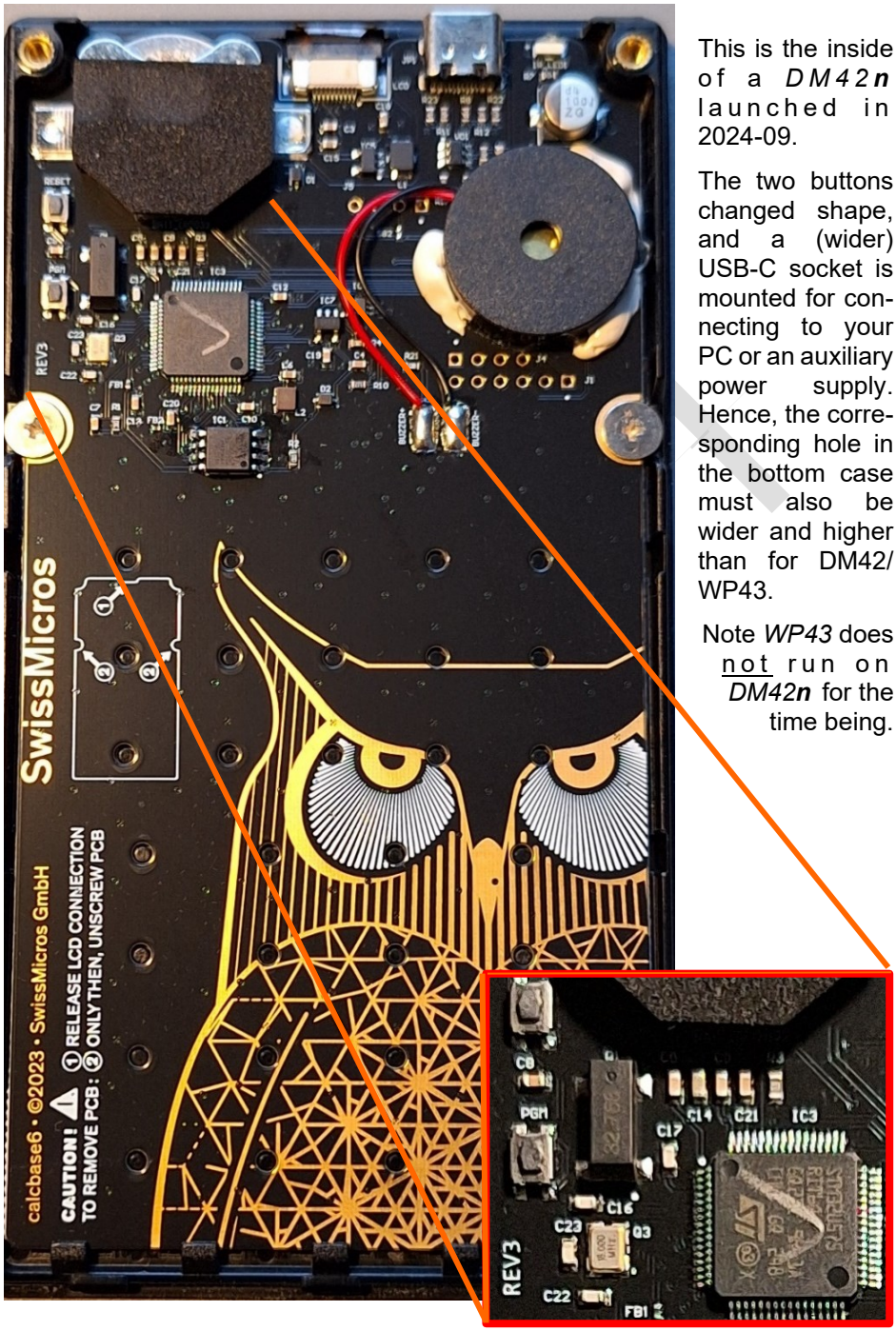
The following link will beam you to a discussion of the various hardware components used on a *DM42 PCB* of 2018-02



as pictured below: *https://www.hpmuseum.org/forum/thread-10143.html* .

See here the *PCB* of an early *DM42* (*β*) of spring 2017. RESET is the top left button. The wide trace around the *PCB* is on minus (−).





This is the inside of a *DM42**n*** launched in 2024-09.

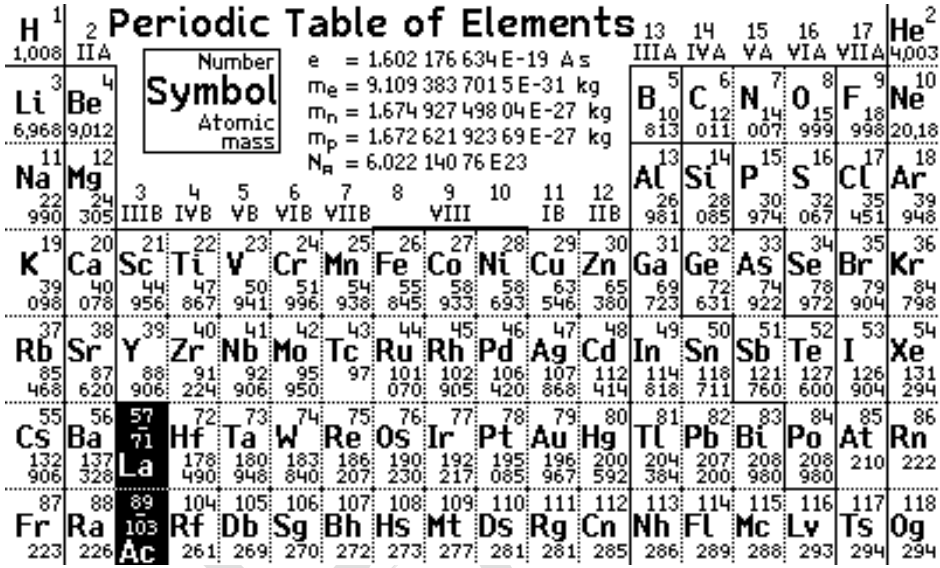
The two buttons changed shape, and a (wider) USB-C socket is mounted for connecting to your PC or an auxiliary power supply. Hence, the corresponding hole in the bottom case must also be wider and higher than for DM42/ WP43.

Note *WP43* does not run on *DM42**n*** for the time being.

Off-Images

When turned off, your *WP43* will show an image until it is turned on again. You can create such *off-images* for your *WP43* yourself. They shall be stored on the *FD* in directory `OFFIMG` – your *WP43* will display them one after the other.

See below and on the frontpage of this manual and the *OM* what could be done:



You can find many examples more as well as how to create such *off-images* at *https://forum.swissmicros.com/viewforum.php?f=14*.

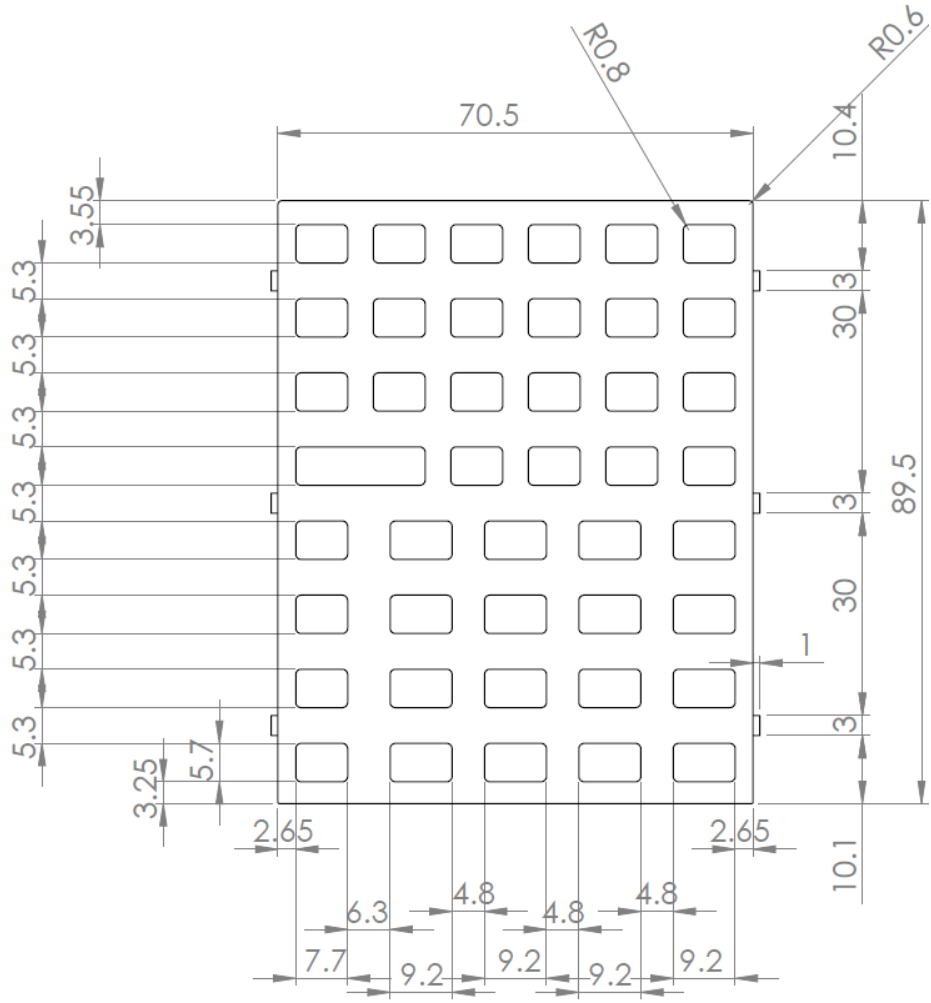
Overlays

A drawing for a blank overlay is printed overleaf, copied directly from *SM* (see *http://technical.swissmicros.com/graphics/key_overlay_DM42.PDF*). All dimensions are given in *millimeters* without tolerances there. This drawing is not printed to scale. Note the overlay is 72.5 mm wide overall and its upper horizontal inter-key spacing is

$$(70.5 - 2 \times 2.65 - 6 \times 7.7)/5 = 3.8\,.$$

For cutting the key voids, do not expect matching punches being readily available. A standard 6 mm square punch might help considerably (found for

< 20 € here in 2024: https://www.amazon.de/Vierkant-Leder-Handwerk-Punch-Gr%C3%B6%C3%9Fen/dp/B00NVHB134/ref=sr_1_37?__mk_de_DE=%C3%85M%C3%85%C5%BD%C3%95%C3%91&crid=3NQMUMGVB2I04&keywords=stanzwerkzeug+papier+vierkant+6mm&qid=1654787046&sprefix=stanzwerkzeug+papier+vierkant%2Caps%2C73&sr=8-37, though certainly available from less relentless sources as well).



In China, there was a manufacturer cutting overlays out of 0.5 mm plastic sheets in 2023, as pictured overleaf. This matter was discussed in

https://forum.swissmicros.com/viewtopic.php?f=24&t=3442 (the picture is copied from it). In 2024, tabbed (0.45 mm thick) or adhesive overlays are available from *Jeff Roetman* (*USA*, *calc.overlays@gmail.com*). Do not choose any thicker material – I recommend 0.35 mm like the overlays of *HP-48* or *HP-41C* were.

Find the original print of keys and keyplate (to scale) for your *WP43* on p. B-32. On p. B-33 is also a simple table to scale for creating your own overlay. Note that all *SM* portrait pocket calculators (and thus also *WP43*) share the same keyboard and slot geometry.

APPENDIX B: MEMORY MANAGEMENT

Data Types

You know ten *DT's* from *OMS 2*. Some more had to be defined for internal use, e.g.:

- 7-*character* strings for all kinds of *names*, e.g., of commands, *variables*, or *labels*,
- system integers in the range of ± 2 147 483 648 (i.e. 32 *bits*),
- *flag* words for storing 144 (i.e. 112 global plus 32 local) *user flags* and 112 *system flags* in 256 *bits* in total,[111]
- two *DTs* for two kinds of *menus*,
- one *DT* of variable length for storing *configurations* (cf. STOCFG and RCLCFG),
- another *DT* for *expressions* in EQN (see *OMS 4*), and
- two more for program steps and routines (see *OMS 3*).

Generally, data (or objects, outside of routines) can be contained in (*global* or *local*) *RoV's*.

A 1-*block* (= 4 *byte*) *register header* is specified for each of the 112 global *registers*, containing a pointer to the data contained in this *register*[112], the *internal DT* (see overleaf), and some data information.[113]

For each (sub-) routine level, 3 *blocks* are reserved in *RAM* ensuring proper return. For every (sub-) routine requiring ***n*** local *registers*, 1 + ***n*** *blocks* are allocated in addition, with the first *block* for the 32 local *flags*.

An 2-*block named variable header* is specified for each *variable*, containing a pointer to the data contained in this *variable*, the *internal DT*, some data information,[113] a pointer to the *name* of the *variable*, and its length.

---

[111] We need far less than 112 *system* flags so far.

[112] It points to the first memory block number of the *register* content. This allows access to 65 535 *blocks* × 4 *bytes / block* = 262 140 *bytes* = 256 kB minus 4 *bytes*. A special value is reserved for the NULL pointer.

[113] Like display base for a *short integer* (2 … 16), sign for a *long integer* (0: zero, 1: negative, 2: positive), angular unit for an *angle* or *real number* (0: *radian*, 1: *grad* or *gon*, 2: *degree*, 3: *mil*, 4: *sexagesimal degree*, 5: *multiple of π*, 6: no *angle* = plain *real*).

The data themselves are then stored in the following way:

		DT number and meaning	Size [in *blocks* or *bytes*]	
1	0	ℤ *Long integer* [114]	≥ 1 + 1	8 - 1109
2	1	ℝ *Real number* [115] (*real34*)	1 + 4	20
3	2	ℂ *Complex number*, always stored in rectangular notation (*complex34*)	1 + 2 × 4	36
4	1	*Angle* [116]	1 + 4	20
5	3	*Time* [117]	1 + 4	20
6	4	*Date* [118]	1 + 4	20
7	5	*Text string* [119]	$1 + \frac{j+1}{4} + \frac{k}{2}$	8 - 100
8	6	*Real* matrix (of ***n*** rows and ***m*** columns) [120]	1 + ***n*** × ***m*** × 4	≥ 20
9	7	*Complex* matrix (of ***n*** rows and ***m*** columns) [120]	1 + ***n*** × ***m*** × 8	≥ 36

[114] This *DT* is for number theory problems. The header *block* indicates the size (in *blocks*) of the *integer* following. 1 *block* following allows for signed *integers* up to $2^{31} \approx 2 \times 10^9$. Size increment is 1 when required. Maximum *long integer* size is 1108 *bytes* equivalent for 1001 decimal digits. Look up the display limits further below.

[115] Standard *reals* feature 128 *bits* (= 16 *bytes*) and 34-digits precision here. For comparison, *WP 34S* standard was 64 *bits* and 16 digits. See the chapter about *Range and Representation of Real Numbers* below.

[116] A tagged *angle* is stored as a *real number* with a special header as described on previous page.

[117] Internally, a *time* or time *interval* is stored as a *real number* of *seconds*, just with a specific header. Decimal times are displayed in *hours*. This format allows for expressing intervals of some 5×10^{11} *years* with *femtosecond* precision, exceeding requirements in this universe.

[118] Internally, a *date* is stored as *real number* of *seconds* passed since **−4713-01-01 12:00:00** (noon – this is date and time zero for *Julian day number* counting).

[119] … containing ***j*** 1-*byte characters* and ***k*** 2-*byte characters* (with bit 15 always set). The header *block* indicates the total string length ***L*** in *blocks*, and 1 stop *byte* after all the *characters* contains **00**. The minimum increment of ***L*** is 1 *block*.

In programs, *text strings* are shorter: there is just 1 *byte* heading and no trailing **00**. See the chapter about *Program Step Sizes* below.

[120] 2 *bytes* for ***n***, 2 *bytes* for ***m***, then follow the individual matrix elements row by row, taking either 4 or 8 *blocks* each.

DRAFT

		DT number and meaning	Size [in *blocks* or *bytes*]	
10	8	*Short integer* [121]	1 + 4	20
11	9	*Configuration* (as stored by STOCFG)	1 + *M* [122]	
12	5	*Label* (or *name* – up to 7 *characters*) [119]	$1 + \frac{j}{4} + \frac{k}{2}$	2 - 5
13		Constant [123]	1	4
14		*System* and *user flags* (112 + 144 *flags*)	8	32
15		Extended precision *real number* (39 digits, for internal use only, not exposed to the user) [124]	1 + 8	36
16		System integer (for internal use only)	1	4
17		User-created *menu* (limited to 1 *view*)	1 + 18 × 7/2	256
18		Predefined *menu* (featuring ***n*** *views*)	1 + ***n*** × 18 × 7/2	
19		Program step	See pp. 202f	
20		Program, containing ***n*** program steps	1 + *M*	
21		*Expression* (for members of EQN), may be stored as *text string* (*DT 7*) as well	1 + *M*	
22		*Calculator state*	1 + *M*	

The *DTs 7 - 9, 11*, and *19*ff are of 'infinite' size limited by available memory (*M*) only. The size of each individual object is fixed though.

Mathematically, each and every *integer* is included in the set of *reals* which is a subset of the set of *complex numbers* (generally: $\mathbb{N} \subset \mathbb{Z} \subset \mathbb{Q} \subset \mathbb{R} \subset \mathbb{C}$). Thus, **3** CPX? should return **true**. Though we doubt you expected that. Hence, ***x*** CPX? will return only **true** if ***x*** is no *real* object, and ***x*** REAL? will be only **true** if ***x*** has a non-zero fractional part. MATR? will return **true** also

[121] This *DT* is for computer science problems.

[122] The size will vary with the number of user assignments being part of the *configuration* (see *OMS 6*).

[123] A pointer to the constant is sufficient here, regardless of its own precision.

[124] There are even longer (i.e. more precise) *reals* used in internal computations. See further below.

DRAFT

for 1x1 matrices. There may be a command DATYP? returning the *data type* of x.

As mentioned above, any object of any *DT* will take one storage space only, i.e. one *RoV*. In consequence, *RoV* lengths may vary considerably. You do not have to bother – the operating system of your *WP43* will take care of all the necessary administration.

Thus, the amount of *RAM* required for data storage is not fixed. Data and programs allocate their memory from the same large pool – monitoring the free space available by MEM?, you may observe differences following, e.g., changing contents of your *stack*.

What Will Happen when Changing Word Size?

WSIZE affects *short integer* contents of all allocated *stack registers* and L exclusively: Reducing word size will truncate all *short integer* values therein down to the new size. Increasing word size will fill empty bits in the significant side of each *short integer* concerned, up to the new size specified.

> **ATTENTION:** Increasing word size will just add empty bits. Thus, a negative *short integer* will immediately become positive then. There is no automatic sign extension! If you want it, take care of it yourself.

All other memory content of your *WP43* will stay as it is. This differs from the implementation as known from the *HP-16C*, where all registers were remapped on word size changes.

When *short integers* are recalled from memory which are longer than current word size, these *integers* will be truncated during recall. When *short integers* are recalled from memory which are shorter than current word size, these *integers* will be filled with empty bits during recall.

SAVE and LOAD…

SAVE stores the *WP43 state* in the standard backup file on the *FD* in directory SAVFILES (a second SAVE will overwrite the first). I/O SAVEST will store the same data but in directory STATE in a named file specified by you, using *DMCP* menus for this. Residing in *FM*, these *WP43 state files* will survive RESETs, even hard ones by the RESET button, and are battery fail safe.

In detail, the data stored by SAVE and SAVEST are all …

1. equations and programs present in *RAM*,
2. *local registers* and *flags*,
3. *global registers*, *global* and *system flags*,
4. statistical sums (if allocated),
5. *variables* (incl. STATS if allocated),
6. user *menus* as well as MyMENU, MyPFN, and Myα,
7. keyboard assignments,
8. the current values of ADM, D.MAX, ISM, REALDF, and #DEC,
9. the settings of DSTACK, GAP, J/G, BESTF, LOCR, HIDE, RANGE, RM, RSD, TDISP, WSIZE, and of the random number generator.

Note that a *calculator configuration*, on the other hand, encompasses the data of points 6 – 9 and all *system flags* marked light blue in the chapter about them (cf. pp. 127ff). A specific *configuration* alone can only be saved in *RAM*.

The various flavours of LOAD allow for recalling these data, according to your requests, and resuming calculator operation after RESET events. Turn to the *IOI* for more information about these commands.

Range and Representation of Real Numbers

Your *WP43* can calculate with *real numbers* of more than 12 000 orders of magnitude (precisely: for $10^{-6143} \leq |x| < 10^{+6145}$). Its software is based on the *decNumber library* supporting arbitrary precision *BCD* numbers (see p. 324).

Overall accuracy should be equal or better than $\pm 3 \times 10^{-33}$ per operation (see pp. 261ff) except at particularly 'nasty' locations.[125] Check your algorithms carefully to avoid such locations. Note that rounding errors accumulate. In most cases, however, precision of your calculatory results will exceed anything known in the real world nevertheless (cf. f. 73 on p. 124).

[125] E.g., 0.5° [cos] [arccos] 0.5° [−] 0.5° [/] returns 3.6×10^{-31} due to the cosine of a small angle being very close to 1.

Results $|x| < 10^{-6176}$ are set to zero. For results $|x| \geq 10^{6145}$, error 4 or 5 will appear unless SPCRES is set (see *App. C*).

All these effects are caused by the internal representation of *reals*: *Standard floating point numbers* are stored in your *WP43* in sixteen bytes using an internal format coarsely following *decimal128* packed coding, though with some exceptions: [126]

- *Real* zero is stored as *integer* zero, i.e. all bits cleared.
- The *significand*[127] of a *real number* is encoded as pure integer in eleven groups of three digits. Each such group is packed into ten bits straight forward, meaning, e.g., $555_{10} = 10\ 0010\ 1100_2 = 22B_H$ or $999_{10} = 11\ 1110\ 0111_2 = 3E7_H$. So the 33 rightmost decimal digits of the *significand* take its least significant 110 bits. Trailing zeros are omitted, so the *significand* will be right adjusted.
- The most significant (128th) bit carries the sign of the *significand*.
- The remaining 17 bits are used for the exponent and the leftmost digit of the *significand*. Of those 17, the lowest twelve are reserved for the exponent (< 4096). For the top five bits (below the sign bit) it becomes complicated – following the standard *IEEE 754*. If these five bits read…
  - `00ttt`, `01ttt`, or `10ttt` then `ttt` takes the leftmost digit of the *significand* (0 – 7), and the top two bits will be the most significant bits of the exponent;
  - `11uut` then `t` will be added to 1000_2 and the result (8_{10} or 9_{10}) will represent the leftmost digit of the *significand*.

    If `uu` reads 00, 01, or 10_2 then these will represent the two most significant bits of the exponent. If it reads 11_2, there are bit patterns specified for encoding special numbers (see below).

[126] Find out about *decimal128* in https://en.wikipedia.org/wiki/Decimal128_floating-point_format. Our coding comes close to what Prof. *William Kahan* called *quadruple* (*precision*) in his paper titled *'IEEE Standard 754 for Binary Floating-Point Arithmetic'* (https://people.eecs.berkeley.edu/~wkahan/ieee754status/IEEE754.PDF) of 1997.

Note *IEEE 754* was developed in 1985, after most of the calculators mentioned in this manual were launched already. See https://en.wikipedia.org/wiki/Floating-point_arithmetic as a starter, also about floating point numbers in general.

[127] *Significand*, not *mantissa* – the *significand* contains no radix mark.

DRAFT

Thus, the maximum absolute value of the stored exponent is $10\ 1111\ 1111\ 1111_2 = 12\ 287_{10}$. For reasons becoming obvious below, 6176_{10} must be subtracted from the stored value to get the true exponent of the floating point number represented. Thus and since $12287 - 6176 + 34 = 6145$ as well as $-6176 + 34 = -6142$, *DT 2* can support 34-digit numbers within $10^{-6143} \leq |x| < 10^{+6145}$.

Rewarding your patience so far, we will show you some illustrative **examples** of encoding in your *WP43* instead of telling you more theory. *SE* stands for the *stored exponent* in the following:

Floating point number	Hexadecimal value stored	Bottom bits in groups of 10	Top 18 bits in binary notation	*SE*
1.	22 08 00 00 00 00 00 00 00 00 00 00 00 00 00 01		0010 0010 0000 1000 00	6176
−1.	a2 08 00 00 00 00 00 00 00 00 00 00 00 00 00 01		1010 0010 0000 1000 00	6176
111.	22 08 00 00 00 00 00 00 00 00 00 00 00 00 00 6f		0010 0010 0000 1000 00	6176
111.111 ($111\ 111\times10^{-3}$)	22 07 40 00 00 00 00 00 00 00 00 00 00 01 bc 6f	06f 06f	0010 0010 0000 0111 01	6173
−123.000 123 ($-123\ 000\ 123\times10^{-6}$)	a2 06 80 00 00 00 00 00 00 00 00 00 07 b0 00 7b	07b 000 07b	1010 0010 0000 0110 10	6170
9.99×10^{99} (999×10^{97})	22 20 40 00 00 00 00 00 00 00 00 00 00 00 03 e7		0010 0010 0010 0000 01	6273
1×10^{-99}	21 ef 40 00 00 00 00 00 00 00 00 00 00 00 00 01		0010 0001 1110 1111 01	6077
-1×10^{-6143}	80 08 40 00 00 00 00 00 00 00 00 00 00 00 00 01		1000 0000 0000 1000 01	33

DRAFT

You will lose one digit precision if you divide 10^{-6143} by 10 and one more for each such division following. At 10^{-6176}, only one digit will be left in the *significand*, stored as 1.

Divide this by 1.999 999 999 999 999 999 999 999 999 999 999 999 and the result will remain 10^{-6176} in default rounding mode (and in RM 1, 2, 3, and 5, cf. RM in the *IOI*). Divide it by 2 instead and the result will become zero.

Let us look at the upper end of our numeric range now:

Floating point number	Hexadecimal value stored	Bottom bits in groups of 10	Top 18 bits in binary notation	*SE*
9.999 999 999 999 999 999 999 999 999 999 999 ×10^{6144} (= 9 999… ×10^{6110})	77 ff be 7f 9f e7 f9 fe 7f 9f e7 f9 fe 7f 9f e7	9 3e7 3e7 3e7 3e7 3e7 3e7 3e7 3e7 3e7 3e7	0111 0111 1111 1111 10	12 286

This *real number* (featuring 34 times the digit 9) is the maximum which can be keyed in directly. It will be shown as ∞ in *startup default* display format; its full glory will be only unveiled by SHOW (cf. p. 83). The greatest legal *significand* is $9\,999\,999\,999\,999\,999\,999\,999\,999\,999\,999 = 10^{34} - 1$.

Additionally, your *WP43* features three 'special *reals*':

Floating point 'number'	Hexadecimal value stored	Top 8 bits in binary notation
∞	78 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0111 1000
−∞	F8 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	1111 1000
NaN ('Not a Number')	7C 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0111 1100

Neither an exponent nor a *significand* is applicable here.

With SPCRES set, these three 'special *reals*' (∞, −∞, and NaN) will be legal outputs of your *WP43*; ∞ and −∞ may be legal numeric inputs as well. NaN covers poles as well as regions where a function result is not defined at all (see the

DRAFT

corresponding entry in *OMS 5, Constants,* and some examples in next chapter).

As mentioned above, some internal calculations are executed in "*internal high precision*", employing even more digits than 34. Nevertheless, not every 34-digit number displayed on your *WP43* will be true to 34 digits (see pp. 261ff). And errors accumulate as explained in f. 64 on p. 114.

Settings of RM may affect results of high precision calculations.

Limitations

Maximum numeric input for *data type (DT)* …

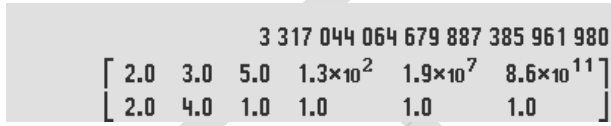
- 1: $\pm 10^{1001}$ or $\pm 2^{3326}$
- 2: ±9.999 999 999 999 999 999 999 999 999 999 999×$10^{RANGE-1}$ (absolute maximum for ***RANGE*** is 6145)
- 3: ±9.999…×$10^{RANGE-1}$ for either part (cf. *DT 2*)
- 4: ±9.999…×$10^{RANGE-1}$ in arbitrary *ADM,* so the actual absolute maximum is 9.999…×$10^{RANGE-1}$ × π.

  ATTENTION: Trigonometric functions actually operate on angles between −**π** and +**π** *radians* exclusively. The precise modulo 2**π** reduction of greater angles ensures that these functions return calculatory correct (though not necessarily meaningful) 34-digit results for an input range of $\pm 10^{999}$ *radians.* Note input $\gtrsim 10^{34}$ for these functions makes no sense since no partial rotation can be specified there anymore.
- 5: $\pm 9\,999\,999\,999\,999\,999\ h\ 59'\,59.999\ldots'' \approx \pm 1.14 \times 10^{12} years$ exceeding the age of the universe.
- 6: 99 999 999 999.123 123 595 9 .d with YMD. Else the input maxima are much smaller: 31.129 999 235 959 .d with DMY and only 12.319 999 235 959 .d with MDY. Minimum is −4 712.010 112 .d with YMD (or −1.014 712 12 .d with DMY/MDY, cf. p. 188) for the first day of year 4712 B.C. 12:00:00 (year 0 for *Julian days*).
- 8: ±9.999…×$10^{RANGE-1}$ for each matrix element (cf. *DT 2*)
- 9: ±9.999…×$10^{RANGE-1}$ for either part of each matrix element (cf. *DT 3*).
- 10: For arbitrary word size n and unsigned mode, $x < 2^n$ (i.e. about 1.8×10^{19} for $n = 64$); for signed modes, $-2^{n-1} < x < 2^{n-1}$.

DRAFT

Internal limits:

- Any operation on long integers must not exceed 3328 bits (corresponding to 6.727×10^{1001}).
- PRIME? works like the other binary tests described in *OMS 3*. Above 3 317 044 064 679 887 385 961 981 ($\approx 3\times10^{24}$), however, NEXTP can find no more primes but 'probable primes' only (cf. p. 71). If you want to apply NEXTP or PRIME? there, consult appropriate reference literature.[128]
- Prime factors will be accurate up to 3 317 044 064 679 887 385 961 980 ($\approx 3\times10^{24}$, cf. p. 41 for FACTOR and see p. 268 for some caveats).



- Any intermediate *real* result within $(-10^{-6176},\ 10^{-6176})$ will be assessed as **0.** (cf. previous chapter). Any intermediate *real* result exceeding $(-10^{6145},\ 10^{6145})$ will be assessed as **−∞** or **∞**, respectively, and an overflow error will be thrown for ~~SPCRES~~. This holds for matrix elements and parts of *complex numbers* as well.
- The equation solver (in EQN) features an independent numeric *stack* (rather than the XYZTABCD *stack*). Both operator and numeric *stacks* are 10-level each, so up to 9 pairs of parentheses may be nested.

Maximum numeric output for *DT* …

- **1:** $\pm10^{1001}$ with full 1001-digit precision (though you can see just up to 298 digits of a very long integer in one view – cf. SHOW on p. 83). DUMP

[128] See *https://en.wikipedia.org/wiki/Miller%E2%80%93Rabin_primality_test* for a start. Whatever is a true prime will be confirmed by PRIME? – a composite may be falsely assessed as prime with a probability of 9×10^{-16}.

With some care and a lot of time (we recommend *USB* power supply), NEXTP on the *WP43* calculator could find 'probable primes' for *long integers* up to some 7.721×10^{317} with few exceptions near the upper limit (hence input is limited to 10^{308} here). You can get a 298-digit prime within some 40 *seconds* on your battery powered *WP43* – and read it entirely on its screen using SHOW.

On the *WP43 Simulator*, NEXTP finds 'probable primes' for *long integer* input up to some 4×10^{1001}; and PRIME? will check them and their neighboring numbers (but you have to use DUMP to get all their digits out). – Presently, it can even do a lot more: if you start with a *real* $>10^{1001}$, NEXTP will return 'probable primes' up to 10^{4000} within some minutes! Although integers $>10^{1001}$ are not supported officially, these numbers can be dumped and are probably correct.

allows to write any number entirely onto the *FD* (cf. p. 37).

- **2, 3, 8, and 9:** The maxima equal those stated for input above. Any *real* result exceeding $(-10^{RANGE},\ +10^{RANGE})$ will be displayed as **-'∞'** or **'∞'**, respectively, and any result within $(-10^{-RANGE},\ +10^{-RANGE})$ or $(-10^{-HIDE},\ +10^{-HIDE})$ will be displayed as **'0.'** as long as ***RANGE*** is within the absolute limits (as stated for intermediate results). Else any *real* result assessed as **0.** will be displayed as such, and anyone assessed as **-∞** or **∞** will be displayed as **-∞** or **∞** if SPCRES is set; else an overflow error will be thrown. This holds for matrix elements and parts of *complex numbers* as well.
- **4:** For angular conversions, the limits are as specified for input above. The functions ARCSIN, ARCCOS, and ARCTAN return values between **-π** and **+π** (or their equivalents) only.
- **5:** The maximum is as specified for input above. Time intervals smaller than the display limit set by TDISP (< 1 ms for TDISP 0) are shown in a format like SCI 4 (or ENG 4 with ALLENG set) but with a trailing 's', e.g., **-1.234 5×10⁻⁹ s** (more digits can be shown by RBR or SHOW).
- **6:** The *date* limits are as specified for input with YMD above. Large *dates* may be displayed as well with DMY or MDY, however; negative *dates* may be displayed like 07.07.-0753.
- **10:** The maxima are as specified for input above.

Further limitations for *data type* …

- **7:** *Text strings* are limited to 196 *characters.*
- **11:** *Configurations* cannot be displayed. Thus, if any *RoV* containing such data is on screen (e.g., in RBR or after RCL), **Configuration data** is shown instead of its content.

Special Results

Throughout this chapter, SPCRES is presumed to be set. Thus, infinities and non-numeric results are legal[129] – no error messages will be thrown if such results happen to occur (cf. p. 194). – The following monadic functions will return either

[129] Results were crosschecked against *WP 34S* wherever possible. Additionally, *WolframAlpha* was used for checking results with finite arguments. Where deviations are observed we are confident your *WP43* returns the correct results.

∞, −∞, or NaN under the conditions stated below:

Input x	Operation(s)	Output for ℝ lit
−1.	artanh	−∞
0 or 0.	ln, lg, lb x	
0.	1/x	∞
1.	artanh	
0 or 0.	Γ(x)	NaN
$\mathrm{Re}(x) < 1$	arcosh	
$\lvert\mathrm{Re}(x)\rvert > 1$	arccos, arcsin, artanh	
±90° or equivalents in other *ADM*	tan	

And the following monadic functions operate also on infinities:

Input x	Operation(s)	Output for ℝ lit
−∞	x^3, $\sqrt[3]{x}$	−∞
	arctan	−90.° or equivalents
	tanh	−1.
	1/x, e^x, 10^x, 2^x, sinc	0.
	x^2, arsinh	∞
−∞	arcosh	NaN
$-\infty \le x < 0$	ln, lg, lb x	NaN
∞	1/x, sinc	0.
	tanh	1.
	arctan	90.° or equivalents
	ln, e^x, x^2, $\sqrt{x}$, lg, 10^x, lb x, x^3, $\sqrt[3]{x}$, sinh, cosh, arsinh, arcosh	∞
−∞ or ∞	cos, sin, tan, artanh	NaN

DRAFT

The functions printed in light yellow fields in the two tables above will also return NaN (or NaN+i×NaN) with *complex* results allowed. Others will change their output when ℂ is lit. – For dyadic functions, we combined the respective tables:

Input y	x	Operation(s)	Output for ℝ lit
∞	$x \neq -\infty$	+	∞ [130]
−∞	$x \neq \infty$		−∞ [130]
−∞	∞	+	NaN [130]
∞	$x \neq \infty$	−	∞ [131]
−∞	$x \neq -\infty$		−∞ [131]
−∞	−∞	−	NaN
∞	∞		
∞	$x > 0$	×	∞ [130]
−∞	$x < 0$		
∞	$x < 0$	×	−∞ [130]
−∞	$x > 0$		
0 or 0.	−∞ or ∞	×	NaN [130]
$0 < y \leq \infty$	0.	/	∞
$-\infty \leq y < 0$			−∞
−∞ or ∞	−∞ or ∞	/	NaN
0 or 0.	0.	/, y^x	
−∞ or ∞	0 or 0.	y^x	
$-\infty \leq y < 0$	−∞ or ∞	$\sqrt[x]{y}$	NaN
$-\infty < y < 0$	non-integer x	y^x, $\sqrt[x]{y}$	NaN
−∞	odd $x > 0$		−∞

[130] Swapping x and y will return the same result here.

[131] Swapping x and y here will return this result times −1.

Input y	x	Operation(s)	Output for ℝ lit
−∞	even $x > 0$	y^x	∞
		$\sqrt[x]{y}$	NaN
$y \neq 0$	−∞	y^x	0.
	∞		∞
0.	$-\infty \leq x < 0$	y^x	∞
	$0 < x \leq \infty$		0.
	$-\infty < x < 0$	$\sqrt[x]{y}$	∞
	$0 \leq x < \infty$		0.
$0 \leq y \leq \infty$	−∞ or ∞	$\sqrt[x]{y}$	1.
∞	$-\infty \leq x < 0$	y^x	0.
	$0 < x \leq \infty$		∞
	$-\infty < x < 0$	$\sqrt[x]{y}$	0.
	$0 \leq x < \infty$		∞
0.	$0 < x < \infty$	$\log_x y$	−∞

Some particular returns of elementary transcendental functions …

- operating at the edge of the *complex* plane at ±∞ (or close to it) or …
- returning values at the edge ...

… are listed below. The left columns contain the input again: [132]

Cartesian Re(x)	Im(x)	Polar r(x)	φ(x)	Op.	Output for ℂ lit
−∞	0	∞	180°	x^2	(∞ ∡ 360° =) ∞+i×0.
0.	∞	∞	90°		(∞ ∡ 180° =) −∞+i×0.
−∞	—	—		$\sqrt{x}$	(∞ ∡ 90° =) 0.+i×∞
−∞	0	∞	180°		

[132] Actually, *complex* infinities should be treated in polar notation (see an article by *HP* in *http://hparchive.com/Journals/HPJ-1984-07.pdf*, p. 27, left column, for reasons).

Cartesian Re(x)	Im(x)	Polar r(x)	φ(x)	Op.	Output for ℂ lit
−∞	—	—		x^3	−∞
−∞	0	∞	180°		−∞+i×0.
−∞	—	—		$\sqrt[3]{x}$	−∞
−∞		∞			−∞ +i×0. [133]
-10^{999}	0	10^{999}	180°		($1.\times10^{333}$ ∡ 60° =) 0.5×10^{333} +i×0.866 025 404×10^{333} $= 5 \times 10^{332}\left(1 + i \times \sqrt{3}\right)$
−∞	0	∞	180°	e^x	0.+i×0.
-10^{999}	10^{999}	10^{999}	135°		0.+i×0.
−∞	∞	∞			NaN+i×NaN
0.	∞	∞	90°		NaN+i×NaN
∞	∞	∞	45°		
∞	0	∞	0°		∞+i×0.
∞	−∞	∞	−45°		NaN+i×NaN
0.	−∞	∞	−90°		
−∞	−∞	∞	−135°		NaN+i×NaN
-10^{999}	-10^{999}	10^{999}			0.+i×0.
−∞	—	—		ln	∞+i×3.141 592 6… $= \infty + i\pi$
−∞	0	∞	180°		
−∞	∞	∞	135°		∞+i×2.356 194 4… $= \infty + i\,{}^{3\pi}/_{4}$
0.	∞	∞	90°		∞+i×1.570 796 3… $= \infty + i\,{}^{\pi}/_{2}$

[133] This result may be calculatory correct but is mathematically incorrect – compare next row. For mathematical reasons, similar cases may occur with other roots or powers operating near the edge of *complex* plane. Note *complex numbers* are generally stored in *Cartesian* coordinates in your *WP43*. Compare previous footnote though.

DRAFT

Cartesian Re(*x*)	Im(*x*)	Polar r(*x*)	φ(*x*)	Op.	Output for ℂ lit
∞	∞	∞	45°		∞+i×0.785 398 1… $= \infty + i\,\pi/4$
∞	—	—			∞
∞	0	∞	0°	ln	∞+i×0.
∞	−∞	∞	−45°		∞−i×0.785 398 1… $= \infty - i\,\pi/4$
0.	−∞	∞	−90°		∞−i×1.570 796 3… $= \infty - i\,\pi/2$
−∞	−∞	∞	−135°		∞−i×2.356 194 4… $= \infty - i\,3\pi/4$
0.	0.	0.	0°		−∞+i×0.
0.	—	—			−∞

Computation of lg and lb x is derived from ln. The same applies in analogy for e^x, 10^x, and 2^x. – At the bottom line, we recommend keeping off ±∞ in *complex* plane.

Program Step Sizes

Program steps are 1 to 3 *bytes* long typically but may be significantly longer. Popular commands (like 1/x, 2^X, ARCCOS, CF, DEC, ENTER↑, FILL, GTO, IP, LN, MOD, NEXTP, RTN, STO, TAN, x^2, y^X, +, $\sqrt{x}$, and almost all binary tests see p. 204) take 1 *byte*, less frequently used ones take 2 *bytes* each. Commands with numeric parameters (like RCL+ 02), lettered *registers* (like x<? Y), or *local labels* (like LBL 55) take one additional *byte* for the parameter; using indirect addressing costs another *byte* more.

Within programs, numeric constants (*reals*) take 4 to 18 *bytes* – see examples below. Maximum length of constants is 4 program lines. Constants recalled from CONST take 2 *bytes* only. *Short integers* take 11 *bytes*, *complex* constants 7 to 34 *bytes*. *Long integers* (like 42) take 4 to 416 *bytes* due to their length.

Global LBL steps specifying an ***n***-letter *label* take ***n*** + 3 *bytes*. Same applies to commands operating on *variables*. Also pure *text strings* in programs consisting of ***n*** *characters* take ***n*** + 3 *bytes*. These statements hold for *characters* with their

DRAFT

Unicode below U+0080$_{16}$ only; any *character* above requires 2 *bytes* (cf. *DT 7* as described on p. 188).

Examples:

`GTO 17` is encoded in 2 *bytes* as follows:
- 1 *byte* for the GTO statement itself;
- 1 *byte* for the *local label* 17.

`RCL+ →X` is encoded in 3 *bytes*:
- 1 *byte* for the RCL+ statement itself;
- 1 *byte* signaling indirect access;
- 1 *byte* for the destination X (= R100).

`LBL 'Prime'` is encoded in 8 *bytes*:
- 1 *byte* for the LBL statement itself;
- 1 *byte* signaling a *global label*;
- 1 *byte* indicating the length of the *label* following (here: 5 usual letters);
- 5 *bytes* for these letters.

`STO 'Count_1'` is encoded in 10 *bytes*:
- 1 *byte* for the STO statement itself;
- 1 *byte* signaling a *variable*;
- 1 *byte* indicating the length of the *variable* following (7 usual letters);
- 7 *bytes* for these letters.

`1e3` is encoded in 6 *bytes*:
- 1 *byte* signaling a numeric constant following;
- 1 *byte* telling its *DT* (short *real number*);
- 1 *byte* indicating its length (3);
- 3 *bytes* for the *characters* constituting the number.

`-1.5` is encoded in 7 *bytes*:
- 1 *byte* signaling a numeric constant following;
- 1 *byte* telling its *DT* (short *real number*);
- 1 *byte* indicating its length (4);
- 4 *bytes* for the *characters*.

$4.160\,643\,081\,028\,110\,401\,606\,014\,040\,140\,401\times10^{-6083}$ (and any similarly big and / or precise *real number*) is encoded in 18 *bytes*:
- 1 *byte* signaling a numeric constant following;
- 1 *byte* telling its *DT* (long *real number*);
- 16 *bytes* = 128 *bits* for the number (cf. p. 191ff for the representation).

DRAFT

8 07 06 05 04 03 02 01$_{16}$ is encoded in 11 *bytes*:

1 *byte* signaling a numeric constant following;
1 *byte* telling its *DT* (*short integer*);
1 *byte* telling its display base (here 16);
8 *bytes* for the digits (any *short integer* ≤ 64 *bits* can be stored therein).

-12 345 is encoded in 9 *bytes*:

1 *byte* signaling a numeric constant following;
1 *byte* telling its *DT* (shorter *long integer*);
1 *byte* indicating its length (6);
6 *bytes* for the sign and digits.

Here are all 122 commands taking 1 *byte* only:

- the four basic arithmetic operators +, -, ×, and /,
- sin, sinh, cos, cosh, tan, tanh, and their inverses,
- the five basic flag commands CF, FC?, FF, FS?, and SF,
- the four basic *Boolean* operations AND, OR, XOR, and NOT,
- RCL, RCL+, RCL-, RCL×, RCL/, STO, STO+, STO-, STO×, and STO/,
- most *stack* commands: DROP↓, ENTER↑, FILL, R↑, R↓, x⇄, and x⇄y,
- all 8 commands of LOOP: DEC, DSE, DSL, DSZ, INC, ISE, ISG, and ISZ,
- 7 angular conversions: D→R, R→D, →DEG, →D.MS, →GRAD, →MULπ, and →RAD,
- 13 basic transcendental functions: 10^x, 2^x, $\sqrt[3]{x}$, e^x, lb, lg, ln, $\log_x y$, x^2, x^3, $\sqrt[x]{y}$, y^x, and $\sqrt{x}$,
- 9 basic programming commands: GTO, INPUT, LBL, PAUSE, ROUND, RTN, STOP, VIEW, and XEQ,
- 1/x, ceil and floor, CLX, COMB and PERM, FP and IP, GCD and LCM, IDIV, max and min, MOD and RMD, NEIGHB, NEXTP, x!, π, +/-, and |x|,
- 28 commands stored in TEST: CONVG?, CPX?, ENTRY?, EVEN?, FC?, FP?, FS?, INT?, KEY?, MATR?, M.SQR?, NaN?, ODD?, PRIME?, REAL?, SPEC?, STRI?, TOP?, ±∞?, and all 9 comparisons with x.

DRAFT

Program Files

Files written by WRITEP are *ASCII* files but not program text, similar to the *.raw* files used on *DM42* (requiring an external converter to and from program text). They employ a binary format similar to the one used in the backup file, rather than a text format due to the constraints of translating all the non-*ASCII characters* used on the *WP43* (but see pp. 334ff). This minimizes the effort in calculator software development and still allows an external converter as for the *DM42*, or an assembler/disassembler as for the *WP 34S* to be developed in a second step. This binary program file format includes the same version numbering as for the backup file.

The *.p43* file format is described at begin of `saveRestorePrograms.c`:

```
// Structure of the program file.
// Format: ASCII
// Each line after line 6 contains the decimal value of a
//                                                  program byte
// +-----+---------------------------+
// | Line| Content                   |
// +-----+---------------------------+
// |  1  | "PROGRAM_FILE_FORMAT"     |
// |  2  | <file format>             |
// |  3  |"WP43_program_file_version"|
// |  4  | <program file version>    |
// |  5  | "PROGRAM"                 |
// |  6  | <program size>            |
// |  7  | <first program byte>      |
// | ... | ...                       |
// |  n  | <last program byte>       |
// +-----+---------------------------+
```

DRAFT

APPENDIX C: MESSAGES AND ERROR CODES

There are some commands generating *temporary information* (*TI*, as specified in *OMS 2*), like CORR, DAY, ERR, L.R., MSG, s, VERS, WDAY, $\bar{x}$, $\hat{x}$, $\hat{y}$, Σ+, Σ−, σ, →POL, →REC, and the binary test commands. Furthermore, the operating system may throw one of the following error messages under the respective conditions (listed alphabetically – *EC* means *error code* here):

Error message	*EC*	Explanations, countermeasures, and examples
An argument exceeds the function domain	1	{1, 2, 3, 4, 10} May be caused by roots of negative numbers or logs for $x \le 0$ (for ~~CPXRES~~), by • 0^0, • $x/0$, $0/0$, • $\Gamma(0)$, • $\tan(\pm 90°)$ and equivalents, • $\operatorname{artanh}(x)$ for $\lvert \operatorname{Re}(x) \rvert \ge 1$, • $\operatorname{arcosh}(x)$ for $\operatorname{Re}(x) < 1$, etc.[134]
Bad time or date input	2	{2, 5, 6} Invalid date format or incorrect *date* or *time* in input, like month > 12, day > 31, or *date-time* exceeding the range limits (cf. *Limitations* on pp. 195ff). Will be thrown as soon as input is closed.
Cannot delete a predefined item	27	{7} Self-explanatory
Cannot read file	35	Futile attempt to LOAD or READP something from *FM*.
Cannot write file	53	Futile attempt to SAVE, SAVEST, or WRITEP to *FM*.

[134] Note that, e.g., tan(90°) and logs of 0 are legal operations on {1, 2, 3} if SPCRES is set. See the end of this appendix.

DRAFT

Error message	*EC*	Explanations, countermeasures, and examples
Distribution parameter out of valid range	16	{1, 2} A parameter specified in **I**, **J**, or **K** is out of valid range for the distribution function called (e.g., LGNRM is called with $j < 0$).
Flash memory is full	23	Regain space by deleting a file from *FM*.
Illegal digit in integer input for this base	9	{10} E.g., 2 in binary or 9 in octal input. Will be thrown as soon as input is closed.
Input data types do not match	31	Attempt to operate on different *DTs* (e.g., for a *Boolean* operation: **10.** AND **11**).
Invalid input data type for this operation	24	Convert what is necessary, if possible (see also **Operation is undefined in this mode**). But this error may also appear in attempts to calculate with *configuration* data.
Invalid name	48	Attempt to create a *variable name* which does not comply to the rules.
Invalid or corrupted data	18	Thrown when there is a checksum error in *FM* or a *FM* segment is not usable otherwise.
Item to be coded	29	May be thrown during firmware development.
I/O error	17	See *OMS 3*.
Large Δ and opposite signs, may be a pole	41	The *Solver* can't find a root but diverging points of opposite sign. This may point to a pole.
Matrix mismatch	21	{8, 9} • A matrix is not square although it should be. • Matrix sizes are not miscible.
No errors calculable for this fit model	40	Will be thrown by the command s(a) if a model except LINF, EXPF, POWERF, LOGF, or ORTHOF was chosen.
No global label in this program	51	Self-explanatory.

DRAFT

Error message	*EC*	Explanations, countermeasures, and examples
No matrix indexed	38	Returned by RCLEL, RCLIJ, STOEL, or STOIJ if there is no *indexed matrix*.
No root found	20	The *Solver* did not converge. Check, e.g., the *HP-34C OHPG* for advice.
No program specified	54	… for solving or integrating.
No statistic data present	28	Attempt to address statistics storage not allocated yet.
No such function	7	Thrown when calling a nonexistent function, e.g., via [XEQ] [α] ... [ENTER↑] (check for typos!). May also be found in equations or after coding on *PC*, where you are free to enter anything – and your *WP43* can find errors no earlier than when executing your code.
No such label found	6	Attempt to address an undefined *label*.
Non-programmable command, please remove	50	Self-explanatory. Such commands can't be entered in *PEM* on your *WP43* but may have slipped into a routine otherwise, e.g., when coding on *PC*.
Not enough memory for such a matrix	39	Will be thrown by M.NEW or M.DIM if allocation of a too large matrix is attempted – or if space is insufficient for saving or recalling STATS.
Not enough memory to load the equations	57	Will be thrown by LOAD or LOADEQ if RAM doesn't suffice.
Not enough memory to load the programs	55	
Operation aborted	56	… by pressing [EXIT] while FACTOR, NEXTP, the *Integrator*, or the *Solver* were running.

DRAFT

Error message	*EC*	Explanations, countermeasures, and examples
Operation is undefined in this mode	13	Caused, e.g., by calling a *real-number* operation in *AIM*. Cf. also **Illegal input data type for this operation**.
Out of range	8	{1, 2, 3, 10} • A number exceeds the valid range. This can be caused by indirectly specifying *word size* > 64 etc. • A *register* or *flag* address exceeds the valid range of currently allocated *registers* or *flags*. May also happen in indirect addressing or when calling nonexistent local addresses. • An R-operation (like R-COPY) attempts accessing invalid register addresses.
Output would exceed 196 characters	33	{7} Maximum length of *text strings*.
Overflow at +∞	4	{1, 2, 3, 8, 9} unless SPCRES is set • Division of a positive number by 0. • Divergent sum, product, integral, derivative. • Positive overflow (cf. pp. 191ff).
Overflow at -∞	5	{1, 2, 3, 8, 9} unless SPCRES is set • Division of a negative number by 0. • Divergent sum, product, integral, derivative. • Negative overflow (cf. pp. 191ff).
Please enter a NEW name	26	Attempt to define a new *variable* or user *menu* taking a *name* already in use (*name* conflict).
RAM does not suffice	11	… to provide the space required. May be caused by attempts to load or write too big routines or matrices, allocate too many *variables*, etc. (cf. pp. 187ff for space needed for various *DTs*). May also happen in program execution due to dynamic allocations (see *OMS 3*).

DRAFT

Error message	*EC*	Explanations, countermeasures, and examples
Singular matrix	22	{8, 9} • Attempt to use an *LU* decomposed matrix for solving a system of equations. • Attempt to invert a matrix which isn't of full rank (a.k.a. irregular or degenerate matrix).
Solver reached a local extremum, no root	42	The *Solver* can't find a root since it got trapped at a local minimum or maximum.
Stack clash	12	STOSTK or RCLSTK attempts accessing *registers* which would overlap the *stack* (see *OMS 1*). Will happen, e.g., with SSIZE8 set and STOSTK **93**.
Syntax error in this equation	45	An equation entered in EQN contains, e.g., implicit multiplication or parentheses not matching or unallowed *characters*.
The function values look constant	44	The *Solver* can't find a root since the function seems to be constant (and never crosses zero).
This does not work with an empty string	34	{7} Self-explanatory.
This equation formula is too complex	46	… since it exceeds 9 levels (parentheses, pending operations).
This item cannot be assigned here	47	You can assign an *item* to MyMENU, MyPFN, Myα, or to a user-created *menu* only. But you cannot assign new *submenus* to any of these.
This system flag is write protected	32	Self-explanatory.
This variable is write protected	37	

DRAFT

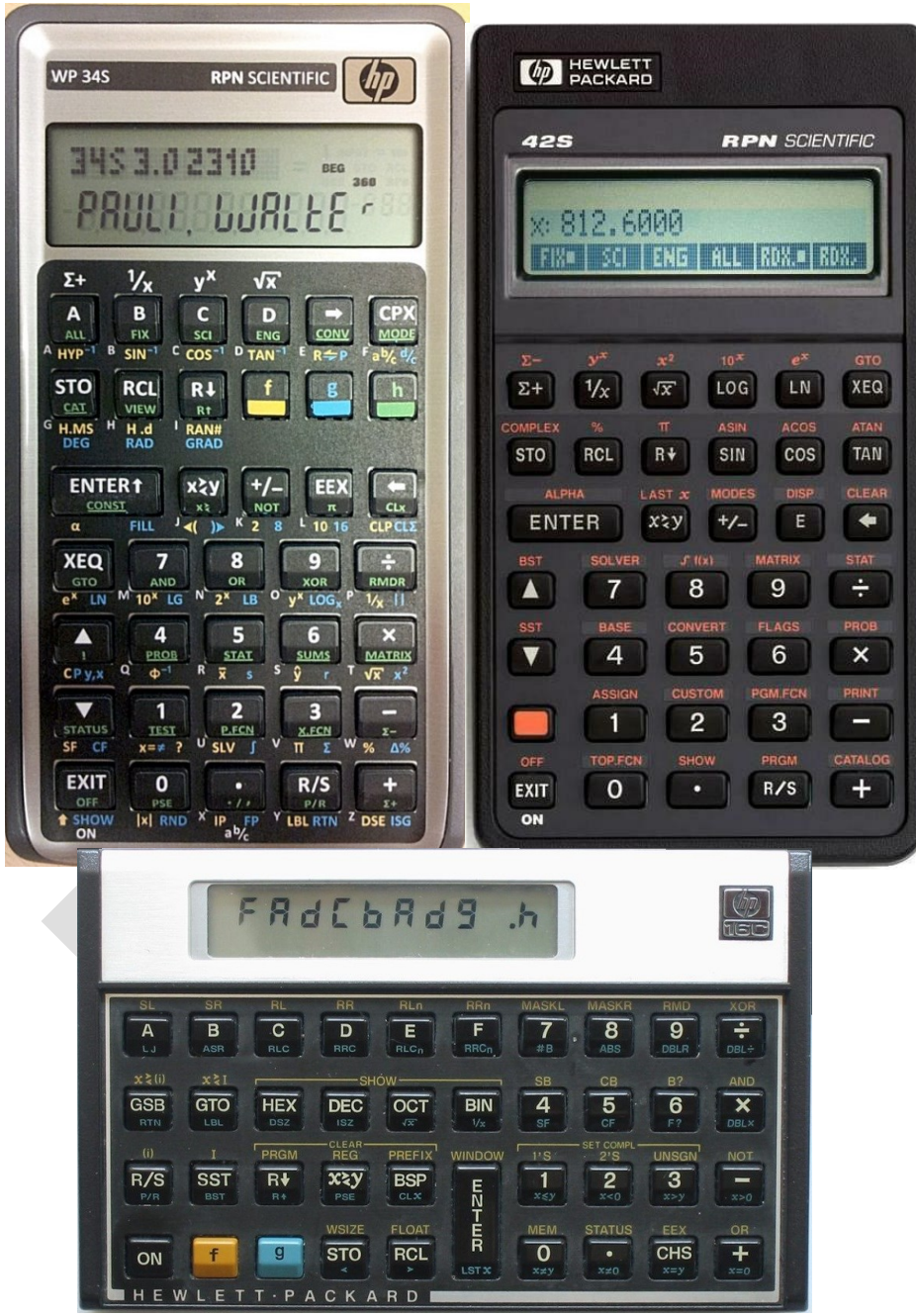
Error message	*EC*	Explanations, countermeasures, and examples
Too few data points for this statistic	15	A statistical calculation was attempted with too few data, e.g., mean or *SD* for < 2 points, regression for < 2 or 3, or s_{mi} for < 30 points.
Too many variables	49	Attempt to create a 1001st *variable*.
Undefined op-code	3	An instruction with an undefined operation code occurred. (Should never happen.)
Undefined source variable	36	Attempt to recall a *variable* not defined yet. (This error should not occur anymore since such a *variable* should be created automatically.)
Word size is too small to take this	14	{10} User input or *register* content is too great to be handled by the *word* size currently set.
≥ 1 initial guess lies out of the domain	43	*Solver* error: One or both initial guesses lie out of the function domain.
	25	Left unused for *WP 34S* compatibility
	10, 19, 30	Left unused since employed earlier in this project

If SPCRES is set, errors 4 and 5 will not occur at all, and error 1 will happen less frequently, since ±∞ and NaN are legal results then (cf. the corresponding entries in CONST on pp. 113ff and the table on p. 194). E.g., [0] [In] will return −∞ then.

Each error message will be displayed in **Z** numeric row and is *temporary information* (see *OMS 2*). So [←], [EXIT], [f] [f], or [g] [g] will erase it and allow continuation most easily. Any other key pressed will erase the message as well, but will also – if applicable – execute with the *stack* contents present.

There are at least two files in *WP43* which may send *TI* to the screen looking like error messages: `display.c` and `screen.c`. Such *TI* will appear in the bottom row (while true error messages show up in row *Z* – cf. *OMS 2*), however, and should not occur in regular operation. If you observe one, please tell us.

DRAFT


WP 34S
RPN SCIENTIFIC
PAUL WALLER
HEWLETT PACKARD
42S
RPN SCIENTIFIC
x: 812.6000
FIX SCI ENG ALL RDX. RDX,
16C
FAdCbAd9 .h
HEWLETT·PACKARD

APPENDIX D: COMPARISON TO THE FUNCTION SETS OF *HP-42S*, *HP-16C*, *HP-21S*, AND *WP 34S*

In the *IOI*, the corresponding functions of vintage *HP* calculators were mentioned under the respective entry of your *WP43*. The tables below revert this view. The first table shows the functions of the *HP-42S* and the corresponding ones of your *WP43* unless they carry identical names and are either both keyboard accessible or both stored in a *catalog* or *menu*. There is an analog table for the *HP-16C* functions starting on p. 219, one for the *HP-21S* on p. 221, and another one for the *WP 34S* on p. 223. Functions newly introduced with *WP* calculators are compiled on pp. 227ff.

Functional differences of homonymous commands are covered in the *IOI* (pp. 14ff).

Corresponding Operations on *HP-42S*

Remarks printed on light grey indicate commands being either default settings or keyboard accessible on your *WP43* while you must use a menu on the *HP-42S*.

HP-42S	*WP43*	Remarks
ACOSH	**arcosh**	In EXP and TRI
ADV	**ADV**	In PRINT (☞)
AIP	Dispensable	Use [+] to merge text and numeric data easily (see *OMS 2*).
ALENG	**αLENG**	In α.FN
ALLΣ	Dispensable	Your *WP43* runs in ALLΣ mode always.
[ALPHA]	[α]	See the description of *AIM* in *OMS 2*.
AOFF	**CF ALPHA**	
AON	**SF ALPHA**	
ARCL	Dispensable	Every *RoV* can hold a *text string*. Use [RCL].
AROT	**αRL** or **αRR**	In α.FN

DRAFT

HP-42S	*WP43*	Remarks
ASHF	**αSL** or **αSR**	In α.FN
ASINH	**arsinh**	In EXP and TRI
ASTO	Dispensable	Every *RoV* can hold a *text string*. Use [STO].
ATANH	**artanh**	In EXP and TRI
ATOX	**α→x**	In α.FN
[BASE]	[INTS] or [BIT]	
BASE+	Dispensable	Your *WP43* executes these arithmetic commands automatically for *short integer* inputs.
BASE−		
BASE×		
BASE÷		
BASE+/−		
BINM	Dispensable	Press [#] [2] for converting any closed *integer number* or integer part in x to binary.
BIT?	**BS?**	In BIT
[BST]	[≡▲] ([▲])	The shortcut works if no *multi-view menu* is open.
CLA	**0** [STO] [K]	
CLD	Dispensable	Any keystroke will clear *temporary information*.
[CLEAR]	[CLR]	
CLRG	**CLREGS**	In CLR
CLST	**CLSTK**	Press [0] [FILL] in *RUM*.
CLV	See remark	*Variables* are cleared or deleted as specified in *OMS 6*.
[COMPLEX]	[CC]	You can also enter *complex numbers* directly using [CC] – see *OMS 2*.
[CONVERT]	[x⇄] & [PART]	
[CUSTOM]	n/a	Create as many *user menus* as memory will hold – not only one CUSTOM menu. Cf. *OMS 6*.

HP-42S	*WP43*	Remarks
DECM	Dispensable	Every input featuring one [.] or [E] is interpreted as a *real* (decimal) number.
DEL	Not featured.	Too dangerous, in our opinion.
DELAY	🖷DLAY	In PRINT (☁)
DELR	M.DELR	In MATX
DET	|M|	
DIM	M.DIM	
DIM?	M.DIM?	
EDIT	M.EDI	
EDITN	M.EDIN	
FCSTX	$\hat{x}$	In FIT
FCSTY	$\hat{y}$	
FNRM	ENORM	In MATX. Euclid is older than Frobenius.
GAMMA	Γ(x)	In PROB; can be called via [x!] as well.
GETKEY	KEY?	In P.FN
GETM	M.GET	In MATX
GROW	M.GROW	
HEXM	Dispensable	Press [#] [H] for converting any closed *integer number* or integer part in x to hexadecimal.
H.MS+	Dispensable	Press [+] or [−] with sexagesimal *times* in x and y.
H.MS−		
INSR	M.INSR	In MATX
INTEG	∫	In ADV
INVRT	[M]$^{-1}$	In MATX
KEYASN	Dispensable	Obsolete since no CUSTOM menu is featured (cf. CUSTOM).

DRAFT

HP-42S	*WP43*	Remarks
LASTx	RCL L	
LBL	LBL	
LCLBL	Dispensable	Obsolete since no CUSTOM menu is featured (cf. CUSTOM). Nevertheless, your *WP43* provides *local labels* (see *OMS 3*).
LINΣ	Dispensable	Your *WP43* runs in ALLΣ mode always.
LIST	n/a	Use PROG instead.
LOG	lg	
MAN	CF T	Manual print mode is *startup default* here.
MAT?	MATR?	In TEST
MEAN	x̄	In STAT
MOD	MOD	
MODES	MODE	
N!	x!	
NEWMAT	M.NEW	In MATX
NORM	Not featured.	
OCTM	Dispensable	Press # 8 for converting any closed *integer number* or integer part in ***x*** to octal.
ON	n/a	Programmable ON is not featured.
PGM.FCN	P.FN	GTO, LBL, RTN, VIEW are on the keyboard.
POSA	αPOS	In α.FN
PRA	Pr K	In PRINT (⎙)
PRGM	P/R	
PRLCD	SNAP	SNAP dumps the entire *LCD* onto the *FD*. The *LCD* of your *WP43* is too large to be printed on either *HP* printer.
PROFF	CF T	

HP-42S	*WP43*	Remarks
PRON	SF T	
PRP	PROG	In PRINT (☁)
PRSTK	STK	
PRUSR	USER	In PRINT (☁)
PRV	r	
PRX	x	
PRΣ	Σ	
PUTM	M.PUT	In MATX
PWRF	PowerF	In FIT
RAN	RAN#	In PROB
RND	ROUND	In PART
RNRM	RNORM	In MATX
ROTXY	RL, RLC, RR, and RRC	In BIT
RTN	RTN	
SDEV	s	In STAT
SIZE	Dispensable	There are 100 numbered global *GP registers* available always – plus up to 99 local ones.
SLOPE	L.R.	In FIT
SST	≡∇ (▼)	Shortcut works if no *multi-view menu* is open.
STR?	STRI?	In TEST
TOP.FCN	Dispensable	… since no top functions can be overwritten.
TRACE	SF T	
TRANS	[M]^T	In MATX
UVEC	UNITV	In MATX and CPX
VARMENU	VARMNU	Truncated to 6 *characters* to fit the *menu* space.

DRAFT

HP-42S	*WP43*	Remarks
VIEW	VIEW	
WMEAN	$\bar{x}_w$	In STAT
WRAP	M.WRAP	In MATX
XTOA	x→α	The conversion is done in **X**.
X<0?, X<Y?	x<?	In TEST
X≤0?, X≤Y?	x≤?	
X=0?, X=Y?	x=?	
X≠0?, X≠Y?	x≠?	
X≥0?, X≥Y?	x≥?	
X>0?, X>Y?	x>?	
YINT	L.R.	In FIT
ΣREG	Dispensable	Statistical summation *registers* on your WP43 do not overlap with *GP registers*.
ΣREG?		
→DEC	→INT 10 or →DEC	# D ... for closed ***x***.
→HR		.d
→H.MS		h.ms
→OCT	→INT 8 or →OCT	# 8
→POL	→P	
→REC		R←
%CH	Δ%	
÷	/	*ISO 80000-2*: "The symbol ÷ should not be used."

DRAFT

Corresponding Operations on *HP-16C*

The table for the functions of the *HP-16C* is sorted following its keyboard layout, starting top left. As for the *HP-42S*, only functions carrying different names on both calculators are listed.

HP-16C	*WP43*	Remarks
RL, RLn	RL	In BIT
RR, RRn	RR	
RLC, RLCn	RLC	
RRC, RRCn	RRC	
÷	/	(see also *ISO 80000-2*: "The symbol ÷ should not be used.")
DBL÷	DBL/	In INTS
x≷(i)	Dispensable	Every *register* may be used for indirection.
x≷I		
SHOW HEX	n/a	Just convert instead.
SHOW DEC		
SHOW OCT		
SHOW BIN		
B?	BS?	In BIT
GSB	XEQ	
HEX	# H or →HEX	
DEC	# D or →DEC	
OCT	# 8 or →OCT	
BIN	# 2 or →BIN	
SF 3, CF 3	SF L, CF L	Leading zeros. Unshifted SF and CF live in FLAG and MODE.
SF 4, CF 4	SF C, CF C	*Carry*.
SF 5, CF 5	SF B, CF B	*Overflow*.

DRAFT

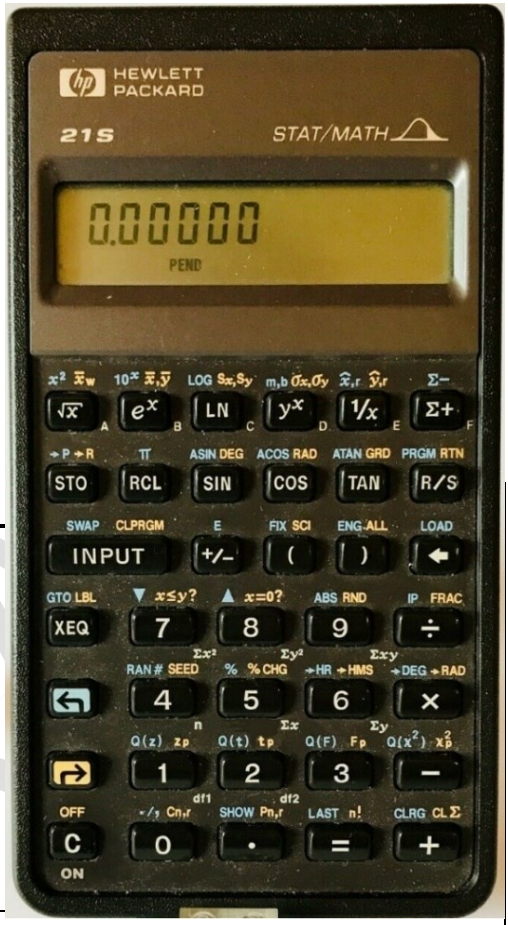
HP-16C	*WP43*	Remarks
F?	FS?	In FLAG
(i)	Dispensable	Every *RoV* may be used for indirection.
I		
CLEAR PRGM	CLP	In CLR. Note here is also CLPALL.
CLEAR REG	CLREGS	In CLR
CLEAR PREFIX	Dispensable	f f or g g will do.
WINDOW	Dispensable	All 64 *bits* can be displayed in one row.
SET COMPL 1S	1COMPL	In MODE and BIT. Note here is also SIGNMT.
SET COMPL 2S	2COMPL	
SET COMPL UNSGN	UNSIGN	
SST	(▼)	▼ works if no *multi-view menu* is open.
BSP	←	
BST	(▲)	▲ works if no *multi-view menu* is open.
x≤y	x≤?	In TEST. Note far more tests are featured on your *WP43*.
x<0	x<?	
x>y, x>0	x>?	
FLOAT	.d	
MEM	STATUS	In FLAG
CHS	+/-	
◀, ▶	Dispensable	All 64 *bits* can be displayed in one row.
LSTX	RCL L	
x≠y, x≠0	x≠?	In TEST. Note far more tests are featured on your *WP43*.
x=y, x=0	x=?	

DRAFT

Corresponding Operations on HP-21S

The table for the functions of *HP-21S* follows the same rules as the one for *HP-16C*. It is, however, an algebraic calculator; hence its keys INPUT, (,), and = have no direct equivalent on your *WP43*.

Consult the *HP-21S OM* for additional information about the four most important continuous statistical distributions and their applications.



HP-21S	*WP43*	Remarks
$\bar{x}_w$	$\bar{x}_w$	In STAT or FIT
$\bar{x}$,$\bar{y}$	$\bar{x}$	
s_x,s_y	s	
m,b	L.R.	
σ_x,σ_y	σ	
$\hat{x}$,r	r, $\hat{x}$	
$\hat{y}$,r	r, $\hat{y}$	
PRGM	P/R	
SWAP	x⇄y	
CLPRGM	CLP	In CLR
INPUT	Dispensable	This functionality is contained in ENTER↑. Also your *WP43* features a command called INPUT but this works in programs.
(,)	Dispensable	You can forget those keys in *RPN*.
LOAD	n/a	Loads predefined programs in the *HP-21S*.

DRAFT

HP-21S	*WP43*	Remarks
ABS	|x|	
RND	ROUND	In DISP
FRAC	FP	In PART
÷	/	Cf. *ISO 80000-2*: "The symbol ÷ should not be used."
SEED	SEED	In PROB
%CHG	Δ%	
Q(z)	$Norml_{\Delta}$	In *submenus* of PROB. Note your *WP43* features the *normal distribution* for arbitrary μ and σ instead of the *standardized* one ($\mu=0$, $\sigma=1$) provided in the *HP-21S*. And the implementation of the inverse distribution functions (a.k.a. quantile functions) deviates on both calculators: your *WP43* calculates with the probability *P* while the *HP-21S* calculates with the error probability $Q = 1 - P$ as input (cf. *OMS 2*). Labeling these functions using the letter *p* on the *HP-21S* may add some confusion.
zp	$Norml^{-1}$	
Q(t)	$t_{\Delta}(x)$	
tp	$t^{-1}(p)$	
Q(F)	$F_{\Delta}(x)$	
Fp	$F^{-1}(p)$	
Q(χ²)	$\chi^2{}_{\Delta}(x)$	
(χ²p)	$(\chi^2)^{-1}$	
Cn,r	COMB	In PROB
Pn,r	PERM	
LAST	RCL L	
=	Dispensable	You can forget that key in *RPN*.
n!	x!	
CLRG	CLREGS	In CLR

DRAFT

Corresponding Operations on *WP 34S*

WP 34S and *WP43* share over 90% of their function sets. It was our objective that your *WP43* shall equal or exceed the performance of *WP 34S* in every aspect. Most of the discrepancies between both calculators are caused by their different displays: your *WP43* allows for *softkeys* – *WP 34S* can only carry four *hotkeys* instead. Also dealing with matrices is distinctly eased by the large high resolution *LCD* of your *WP43*; thus some elementary matrix commands of *WP 34S* are not required anymore on your *WP43*.

Remarks printed on light grey indicate commands being either default settings or obsolete on your *WP43* while you have to use them on the *WP 34S*.

WP 34S	*WP43*	Remarks
ANGLE	∠	
Binom	Binom▲	
Binom$_u$	Binom▲	
Cauch	Cauch▲	
Cauch$_u$	Cauch▲	
CLα	0 STO K	Check the *OM* for the conditions when this *register* is used.
CONV	U→	
DBLOFF	Dispensable	Your *WP43* features *34-digit DTs* per default – no need for a *double precision* mode anymore.
DBLON		
Expon	Expon▲	
Expon$_u$	Expon▲	
F(x)	F▲(x)	
F$_u$(x)	F▲(x)	
dRCL	Dispensable	Cf. DBLOFF.

DRAFT

WP 34S	*WP43*	Remarks
gCLR, gDIM, gDIM?, gFLP, gPIX?, gPLOT, gSET	n/a	The *LCD* of your *WP43* features 240 × 400 px compared to 6 × 43 px of *HP-30b* – the graphic paradigm of *WP 34S* makes no sense on your *WP43* anymore. On the other hand, it was not our objective designing a graphing calculator. Thus, we include just the basic graphic support of *HP-42S* plus ASSESS, CENTRL, HIST, POINT, and PLOT.
Geom	**Geom**	
Geom$_u$	**Geom**	
GTOα	Dispensable	Use GTO with an appropriate parameter instead.
H.MS+, H.MS−	Dispensable	Your *WP43* features a dedicated *DT* for *times*, so + and − suffice for adding and subtracting sexagesimal *times*.
INTM?	Dispensable	Your *WP43* features two dedicated *DTs* for *integers* – no integer mode needed anymore.
iRCL	Dispensable	Your *WP43* features various *DTs*.
I_x	**I_{xyz}**	This is a triadic function after all.
Lgnrm	**LgNrm**	
Lgnrm$_u$	**LgNrm**	
L_n	**L_m**	Renamed to avoid a search conflict with **ln**.
$L_{nα}$	**$L_{mα}$**	Renamed in consequence to **L_m**.
Logis	**Logis**	
Logis$_u$	**Logis**	
MROW+×, MROW×	Dispensable	Obsolete matrix commands.
MROW⇆	**M.R⇆R**	
M+×	Dispensable	Obsolete matrix command.
M^{-1}	**$[M]^{-1}$**	

DRAFT

WP 34S	*WP43*	Remarks
M-ALL, M-COL, M-DIAG, M-ROW	Dispensable	Obsolete matrix commands.
M×	Dispensable	Your *WP43* features two dedicated *DTs* for matrices. Thus, you can multiply and copy matrices just like any other objects.
M.COPY		
M.IJ, M.REG	Dispensable	Obsolete matrix commands.
nBITS	**#B**	
nCOL, nROW	Dispensable	Obsolete matrix commands.
Norml	**Norml**$_{\blacktriangle}$	
Norml_u	**Norml**$_{\blacktriangle}$	
Poiss	**Poiss**$_{\blacktriangle}$	
Poiss_u	**Poiss**$_{\blacktriangle}$	
REALM?	Dispensable	Your *WP43* features a dedicated *DT* for *real numbers* – no need for a *real* mode anymore.
RECV	Replaced by **READP**	Exchanging programs via the *FD* is easier.
REGS, REGS?	Dispensable	There are 100 numbered global *GP registers* on your *WP43* always.
SENDA, SENDP, SENDR, SENDΣ	Replaced by **WRITEP**	Exchanging programs via the *FD* is easier.
SEPOFF, SEPON	**GAP**	
SHOW	**RBR**	
sRCL	Dispensable	Cf. DBLOFF.

DRAFT

WP 34S	*WP43*	Remarks
TRANSP	[M]^T	
TSOFF	GAP 0	
TSON	GAP 3	
t(x)	$t_{\blacktriangle}(x)$	
t_u(x)	$t_{\blacktriangle}(x)$	
VIEWα, VWα+	Dispensable	Use VIEW or AVIEW instead; *text strings* are just another *DT*. Combine text and numeric data easily as shown in *OMS 2.*
Weibl	$Weibl_{\blacktriangle}$	
$Weibl_u$	$Weibl_{\blacktriangle}$	
XEQα	Dispensable	Use XEQ with an appropriate parameter instead.
XTAL?	Dispensable	A quartz crystal is installed by default.
YDOFF, YDON	Dispensable	Your *WP43* displays y whenever requested and possible. Cf. DSTACK.
αDATE, αDAY	Dispensable	You can combine text and numeric data easily as shown in *OMS 2.*
αGTO	Dispensable	Use GTO with an appropriate parameter instead.
αIP, αMONTH	Dispensable	Cf. αDATE.
αRCL, αRC#	Dispensable	Your *WP43* features various *DTs* and 'knows' which type is in the *RoV* specified.
αSTO	Dispensable	Simply press STO instead (every *register* can take a *text string*).
αTIME	Dispensable	Cf. αDATE.
αXEQ	Dispensable	Use XEQ with an appropriate parameter instead.
β	β(x,y)	
Γ	Γ(x)	

WP 34S	*WP43*	Remarks
ΔDAYS	Dispensable	Simply subtract two *dates*.
ζ	ζ(x)	
Φ(x) …	Dispensable	Use NORML… with μ=0 and σ=1 instead.
χ^2(x)	$\chi^2_{\blacktriangle}$(x)	
χ^2_u(x)	$\chi^2_{\blacktriangle}$(x)	
→H	→HR	
🖶PLOT	n/a	Cf. gCLR.
🖶$^C r_{XY}$	Dispensable	Use 🖶r instead.
🖶α, 🖶α+, 🖶+α	Dispensable	Combine text and numeric data as shown in *OMS 2*. Then use 🖶r.
🖶?	Dispensable	A quartz crystal and the proper firmware for printing are installed by default.

New Commands on your *WP43*

The following table lists the commands and pseudo-commands created for your *WP43* (and for preceding *WP* calculators, if applicable), offering new or extended functionality compared to earlier *HP RPN* and algebraic pocket calculators. These are some 390 operations, not counting the unit conversions and constants provided; 127 of them are even new and 19 extended compared to earlier *WP* calculators. The commands are printed below as spelled on your *WP43*. Commands for probability and statistics are printed on light blue background, bit and integer commands on light green.

Command	***WP43***	*WP 31S*	*WP 34S*
2^x AGM	●	—	new
ALL	●	●	extended
AND ASR	●	—	extended

Command	***WP43***	*WP 31S*	*WP 34S*
ASSESS atan2	**new**	—	—
BACK	●	—	new
BATT?	●	●	new
BC?	●	—	new
BestF	extended	●	●
BestF?	**new**	—	—
$Binom_p$ $Binom_{\triangle}$ (of *Binomial distribution*)	●	●	new
B_n B_n^*	●	—	new
BUZZ CALC	**new**	—	—
CASE	●	—	new
CauchF $Cauch_p$ $Cauch_{\triangle}$ $Cauch_{\triangle}$ $Cauch^{-1}$	**new**	—	—
ceil	●	—	new
CENTRL CLBKUP CLCVAR	**new**	—	—
CLFall	●	—	new
CLKEYS CLMall CLMyM CLMyP CLMyα	**new**	—	—
CLPall	●	—	new
CLVall cn(υ,m)	**new**	—	—
conj CONVG? COV	●	—	new
CX→RE	**new**	—	—
DATE	●	—	(●)
DATE→ DAY DEC	●	—	new
DECOMP	●	●	new
DEG→	●	—	new

DRAFT

Command	*WP43*	*WP 31S*	*WP 34S*
DelITM DelMall DelVall DISK? dn(υ,m) Draw	**new**	—	—
DROP↓ (returned after 50 years, see *App. H*)	●	—	back
DROPy DSL DSTACK DUMP	**new**	—	—
D.MS→ D→J	●	—	new
D→MIL EDIT EIGVAL EIGVEC	**new**	—	—
ENTRY?	●	—	new
EQ.DEL EQ.EDI EQ.NEW	**new**	—	—
erf erfc ERR EVEN	●	—	new
$Expon_p$ $Expon_{\triangle}$ $Expon_{\blacktriangle}$ $Expon^{-1}$	●	●	new
EXPT	●	—	new
E(m) E(φ,m) FACTOR	**new**	—	—
FB	●	—	new
FBR	**new**	—	—
FC?F FC?S FF FIB	●	—	new
FILL	●	●	new
floor FP?	●	—	new
$F_p(x)$ $F_{\triangle}(x)$ (of *F distribution*)	●	●	new
FS?F FS?S	●	—	new
F(φ,m)	**new**	—	—
f'(x) f''(x)	extended	—	new
GAP	extended	●	new
GaussF	**new**	—	—
GCD	●	●	new
g_d g_d^{-1} $Geom_p$ $Geom_{\triangle}$ $Geom_{\blacktriangle}$ $Geom^{-1}$ GRAD→ H_n H_{nP}	●	—	new

DRAFT

Command	*WP43*	*WP 31S*	*WP 34S*
HIDE HIDE?	new	—	—
HISTOX HISTOY $Hyper_p$ $Hyper_{\blacktriangle}$ $Hyper_{\blacktriangle}$ $Hyper^{-1}$ HypF			
IDIV	●	—	new
IDIVR	new	—	—
Im			
INC INT? ISE ISM? I_{xyz}	●	—	new
$J_y(x)$ J/G?	new	—	—
J/G	extended	●	new
J→D KEY? KTYP?	●	—	new
K(m)	new	—	—
lb LBL?	●	—	new
LCM	●	●	new
LEAP?	●	—	new
$LgNrm_p$ $LgNrm_{\blacktriangle}$ $LgNrm_{\blacktriangle}$ $LgNrm^{-1}$			
L_m $L_{m\alpha}$ lnβ lnΓ			
LoadAll	●	●	new
LoadEq LoadPr LoadR LoadSS	●	—	new
LoadV	new	—	—
LoadΣ	●	—	new
LocR LocR?			
$Logis_p$ $Logis_{\blacktriangle}$ $Logis_{\blacktriangle}$ $Logis^{-1}$	●	●	new
log_xy	●	—	new
MAD	new	—	—
MANT max	●	—	new
MEDIAN	new	—	—
MIL MIL→ MIL→D MIL→R			

DRAFT

Command	*WP43*	*WP 31S*	*WP 34S*
min MIRROR	●	—	new
MOD	●	●	new
MONTH MSG	●	—	new
MULπ MULπ→ Mπ→R	**new**	—	—
M.LU M.SQR? NAND NaN?	●	—	new
$NBin_p$ $NBin_{\triangle}$ $NBin_{\triangle}$ $NBin^{-1}$	**new**	—	—
NEIGHB	●	—	new
NEXTP	extended	●	new
NOR	●	—	new
$Norml_p$ $Norml_{\triangle}$ $Norml_{\triangle}$ $Norml^{-1}$	●	●	new
NOT	●	—	extended
NXTFIT	**new**	—	—
nΣ (called by name)	●	●	new
ODD?	●	—	new
PLAY PLOT POINT	**new**	—	—
$Poiss_p$ $Poiss_{\triangle}$ $Poiss_{\triangle}$ $Poiss^{-1}$	●	●	new
PopLR	●	—	new
PRIME?	extended	●	new
PUTK	●	—	new
Q_3-Q_1	**new**	—	—
RAD→	●	—	new
RANGE RANGE? RANI#	**new**	—	—

Command	***WP43***	*WP 31S*	*WP 34S*
RBR	●	●	new
RCLCFG	extended	—	new
RCLSTK RCLMAX RCLMIN RDP	●	—	new
Re READP REDRAW RE→CX Re⇄Im	**new**	—	—
RJ	●	—	new
RL RLC	●	—	extended
RM	●	—	new
RMD	●	●	extended
RM?	●	—	new
RootF	**new**	—	—
ROUNDI	●	—	new
RR RRC	●	—	extended
RSD R-CLR R-COPY R-SORT R-SWAP RTN.SKP	●	—	new
R→MIL R→Mπ	**new**	—	—
SAVE	●	●	new
SAVEST SDIGS?	**new**	—	—
SDL SDR SETCHN	●	—	new
SETDAT	●	—	(●)
SETEUR SETIND SETJPN	●	—	new
SETSIG	**new**	—	—
SETTIM	●	—	(●)
SETUK SETUSA SIGNMT sinc	●	—	new
sincπ	**new**	—	—
SKIP	●	—	new

DRAFT

Command	***WP43***	*WP 31S*	*WP 34S*
SL	●	—	extended
SLVC	**new**	—	—
SLVQ	●	—	new
s_m s_{mw}	●	●	new
s_{mi} SNAP sn(υ,m)	**new**	—	—
SPEC?	●	—	new
SR	●	—	extended
SSIZE?	●	●	new
STATUS	extended	—	extended
STOCFG	extended	—	new
STOSTK STOMAX STOMIN	●	—	new
s_w	●	●	new
s_{xy}	●	—	new
s(a) TDISP	**new**	—	—
TICKS	●	—	new
TIME TIMER	●	—	(●)
T_n TOP?	●	—	new
$t_p(x)$ $t_{\blacktriangle}(x)$ (of *t distribution*)	●	●	new
t↕ ULP? U_n	●	—	new
↶ (UNDO)	●	new	—
VERS?	●	●	new
VOL VOL+ VOL- VOL? V↯	**new**	—	—
WDAY $Weibl_p$ $Weibl_{\blacktriangle}$ $Weibl_{\blacktriangle}$ $Weibl^{-1}$ WHO?	●	●	new

Command	***WP43***	*WP 31S*	*WP 34S*
W_m W_p	●	—	new
WRITEP	**new**	—	—
WSIZE? W^{-1} $\bar{x}_G$	●	—	new
$\bar{x}_H$ x_{max} x_{min}	**new**	—	—
XNOR	●	—	new
XOR	●	—	extended
$\bar{x}_{RMS}$ x→DATE	**new**	—	—
x<? x≤? x=? x≠? x≥? x>?	extended	—	extended
x=+0? x=-0? x≈? YEAR	●	—	new
y^x	extended	●	●
$Y_y(x)$	**new**	—	—
Y.MD	●	●	new
y⇄	●	—	new
ZOOM	**new**	—	—
z⇄	●	—	new
αLENG?	extended	—	●
αPOS?	extended	—	—
αRL αRR αSL αSR	●	—	extended
β(x,y) Γ_p Γ_q Γ_{xy} γ_{xy}	●	—	new
Δx	**new**	—	—
ε ε_m ε_p ζ(x)	●	—	new
Z(φ,m)	**new**	—	—
Π_n	●	—	new

DRAFT

Command	*WP43*	*WP 31S*	*WP 34S*
Π(υ,m) Σ^1/x Σ^1/x^2 Σ^1/y Σ^1/y^2	**new**	—	—
$\Sigma\ln^2 x$ $\Sigma\ln^2 y$ $\Sigma\ln x$ $\Sigma\ln xy$ $\Sigma\ln y$ (called by *names*)	●	●	new
Σlny/x	**new**	—	—
Σ_n σ_w	●	—	new
Σx Σx² Σx²y (called by *names*)	●	●	new
$\Sigma x^2\ln y$ $\Sigma x^2/y$ Σx^3 Σx^4	**new**	—	—
Σxlny Σxy (called by *names*)	●	●	new
Σx/y	**new**	—	—
Σy Σy² Σylnx (called by *names*) $\chi^2_P(x)$ $\chi^2_{\triangle}(x)$ (of *chi-square distribution*)	●	●	new
ψ(υ,m)	**new**	—	—
$(-1)^x$	●	—	new
±∞?	**new**	—	—
×MOD ^MOD →DATE	●	—	new
→DEG	●	●	new
→D.MS	**new**	—	—
→GRAD	●	—	new
→INT →MIL →MULπ	**new**	—	—
→RAD	●	●	new
→REAL	**new**	—	—
⋗⋖	●	—	new
||	●	●	new
%ile	**new**	—	—

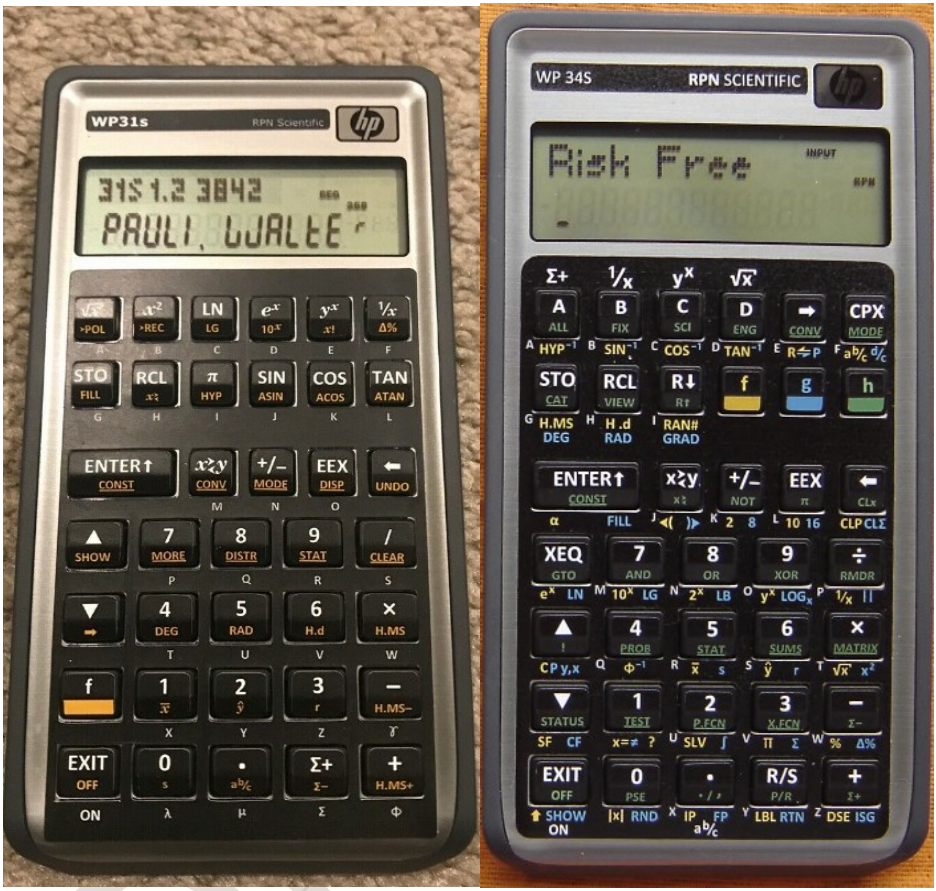
DRAFT

The statements in parentheses in the rightmost column refer to the *WP 34S* with the optional quartz and capacitors installed for an accurate real-time clock (please see its manual). Both a *WP 34S* and a *WP 31S* are pictured overleaf. For creating one for yourself,

1. look for a *HP-20b* or *-30b* (the latter is recommended for featuring a better keyboard),
2. get a programming cable,
3. a file for flashing,
4. software for transmitting said file to the *HP-20b/-30b*, and
5. stickers for bezel and keys.

See the respective manuals for more (note these are under copyright – please respect it). Alternatively, you might get these calculators still custom-made (please search the internet) or find them (used) in one of the usual bays.

DRAFT



Reference Literature

As mentioned above, some advanced functionality of your *WP43* is taken over from previous *HP* calculators. The following vintage *HP* manuals are recommended as source of in-depth information (as far as calculating, programming, and applications are concerned) about the topics listed, from a calculator point of view:

Topic	Recommended literature
Statistical distributions and their application	*HP-21S Owner's Manual*, especially pp. 63 – 105
Manipulating bits and *short integers*	*HP-16C OH*
Programming	*HP-42S OM* *HP-42S PET*
Root finding and numeric integration	*HP-34C OH and Programming Guide* *HP-15C OH* [135] *HP-15C Advanced Functions Handbook* [136] *HP-42S PET*
Accuracy of numeric calculations	*HP-15C Advanced Functions Handbook,* pp. 172 – 211 [136]
Algebraic equation solver	*HP-27S OM*
Financial calculations	*HP-17BII+ User's Guide* [137]

These manuals can be found in the internet – three of them are even provided by *HP* still. A great collection (in every aspect) of literature about *HP* calculators (and related devices) is stored by *Eric Rechlin* at *https://literature.hpcalc.org/* – here you find best quality scans of all the manuals mentioned above (and more), most of them available in various languages. We recommend this site although its handling of copyright shows potential for improvement.

Depending on your education and professional qualification, textbooks about various mathematical, scientific, engineering, or business topics may be helpful in addition. Ensure you know enough about what you compute and how to do it properly (and please watch f. 161 on p. 267, as well as p. 19 of the *OM*).

[135] Download a reprint from *http://h10032.www1.hp.com/ctg/Manual/c03030589.pdf*

[136] Download a reprint from *http://h10032.www1.hp.com/ctg/Manual/c03308725.pdf*

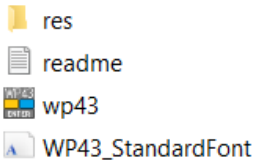
[137] Download a reprint from *http://h10032.www1.hp.com/ctg/Manual/c00363348.pdf*

DRAFT

APPENDIX E: EMULATING A *WP43* ON YOUR *PC*

Under *MS Windows*, you can employ the *WP43 Win Simulator* on your *PC*, featuring almost the same functionality as the calculator. Find it under *https://gitlab.com/rpncalculators/wp43/-/releases*.[138]

You will be asked to open a zipped folder. Extract it to any suitable location on your *PC*. Open its root directory and you will find this:



Unless you did it before, install the font. Start *WP43* by double-clicking on `WP43`. The *Simulator* window will open, looking like the picture here though larger.[139]

Operate the *Simulator* with the mouse. The keys [0] … [9], [.], [ENTER↑], [+], [−], [×], and [/] may be entered directly on the numeric keypad of your *PC*, [▲] and [▼] via the cursor up and down keys, and [←] via [←Backspace]. Further *PC* keyboard shortcuts to *Simulator* keys are presented overleaf.

[138] There are also *Simulators* for *MacOS* and *Linux* available. Please ask on the forum.

[139] A vertical screen size of ≥ 980 px is required, else the landscape window (as shown overleaf) will open, needing 1000 × 568 px.

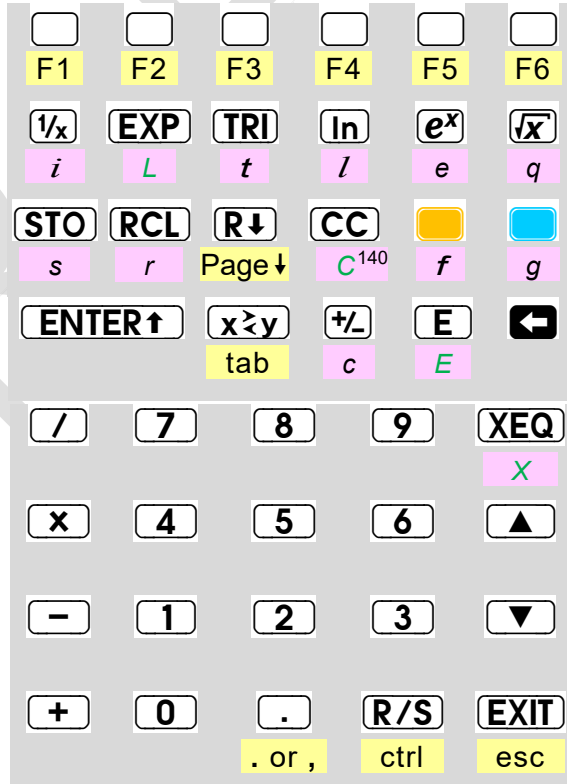
With development ongoing, there may be errors in startup. If you want to ensure the previous *state* will be loaded after an update to continue your work, do as follows:

1. Press [SAVE] in the old version to back up to `wp43.sav`
2. Update the *WP43 Simulator*.
3. If `backup.bin` will not load in the new version automatically, delete it (`rm backup.bin`). Start `wp43`, then use [I/O] **LoadAll** to load `wp43.sav`..

Right clicking with the mouse will call 🟦-shifted labels directly – 🟦EXIT will close the Simulator.

Note that in *AIM* the shortcuts printed on violet will vanish, so you can use your *PC* keyboard for typing *characters* as usual. This may be beneficial in *PEM,* especially.

Taking advantage of this opportunity, your *PC* keyboard settings will apply, e.g., for upper and lower case letters. And "page ⇧" and "page ⇩" will become shortcuts to 🟨 and 🟦, respectively (e.g., for Greek letters).



[140] Capitals are printed in green here. And "ctrl" means the right "ctrl" key.

Furthermore, pressing …

… **F9** copies the precise content of **X** to the clipboard. This is the way to output also very long integers > 10^{298} in their full glory from the *Simulator* (cf. DUMP on p. 37).

… **F10 (Print)** copies the entire 240×400 pixel *Simulator* screen image to the clipboard, e.g., for recording or documentation (cf. SNAP on p. 86).

Beyond *AIM*, pressing …

… ***z*** copies the full contents of all 12 lettered *registers* (**K**, **J**, **I**, **L**, **D**, **C**, **B**, **A**, **T**, **Z**, **Y**, and **X**) to the clipboard.

… ***Z*** copies the full contents of all 112 global *registers* thereto (**K**, **J**, …, **Y**, **X**, **R99**, …, **R00**).

… **²** (i.e. [alt gr] + [2]) inserts x^2.

… **‘** will open *AIM*.

ATTENTION:

While you work with the *Simulator* under *MS Windows*, SAVE will write the backup file to the operating system disk (usually `C:`) into the directory `msys64\home\`***user***`\wp43\SAVFILES` (***user*** stands for your user name).

Calling I/O **SAVEST**, the respective *state file* will go into …`wp43\STATE`.

And WRITEP will write program files into …`wp43\PROGRAMS`.

At the bottom line, whatever your calculator writes to the *FD*, the *Simulator* will write to …`wp43\` into the corresponding subdirectory.

How to Exchange Programs Between Your *WP43* and Your *PC*

I/O **WRITEP** writes a program from program memory (in *RAM*) onto the *FD* in a named file. I/O **READP** reads a named file from the *FD* into program memory.

To transfer a program from your *WP43* to your *PC*,

1. use WRITEP on your *WP43* to write the program onto the *FD* in the directory `PROGRAMS`; WRITEP will ask you for the *source program name*

DRAFT

(i.e. its first global label) and a *destination file name*.

ATTENTION 1: You may choose the *file name* entirely independently of the *program name*. Keep suitable records telling you which file contains which program.

ATTENTION 2: WRITEP will overwrite a file in this directory carrying the same name!

With writing completed, WRITEP will display **Saved**.

2. Connect your *WP43* to your *PC* using a proper *USB* data cable.

3. Press [USB] to 'activate' the connection, so the *FD* will be seen by your *PC*. (If it is not seen by your *PC*, however, then either
   - your cable is no data cable or
   - it does not connect properly or
   - it is defective. Exchange it and return to step 2.)

4. Start the *WP43 Simulator* and

5. call [I/O] **READP** on the *Simulator* to read the program into its memory. READP will display a list of `.p43` files in the default source directory (see previous page) to choose from – you may have to redirect it to `DM42:\PROGRAMS`. With reading completed, READP will display **Program file loaded**.

   The program read in will be the last program in memory; note it may carry the same name as another program being present there already.

   Now you can do with it whatever you like, e.g., debug it by single-stepping through it as explained in *OMS 3*. You can take advantage of your *PC* keyboard whenever you want to change something and *AIM* is active.

   If you expect your *Simulator* session taking more time, you can 'eject' the *FD* from your PC, then press [EXIT] [ENTER↑] on your *WP43*, optionally disconnect the cable, and use your WP43 independently in the meantime.

   Eventually, when the program does what it was meant to do and you have verified everything, you can

DRAFT

return your program from your *PC* to your *WP43* like this:

a. If you had disconnected, connect your *WP43* to your *PC* again (cf. step 2).

b. If your *WP43* is off, turn it on and call [USB] (cf. step 3).

c. Verify that the connection between your *PC* and your *WP43* is active (i.e. the *FD* is seen by your *PC*; cf. step 3).

d. Call WRITEP on the *Simulator* to write your refined program (you will be asked for its *name*) into `DM42:\PROGRAMS`. WRITEP will ask you for a *destination file name* (cf. step 1). With writing completed, WRITEP will display **`Saved`**.

e. Eject the *FD* (logically) from your *PC*.

f. Press [EXIT] [ENTER↑] on your *WP43* calculator to return from active *USB disk* mode (you may disconnect the cable now) and

g. call READP which will display a list of `.p43` files in the directory `PROGRAMS` to choose from. Select your file to read the refined program into the *WP43* program memory. With reading completed, READP will display **`Program file loaded`**. The program read in will be the last program in memory; note it may carry the same name as another program being present there already.

Enjoy!

DRAFT

APPENDIX F: FLASHING & UPDATING YOUR *WP43*

There are two ways to get your hands on a *WP43* calculator:

1. You can buy a *WP43* off the shelf (cf. p. 3) or
2. you can flash a *DM42*.

Way 2 allows you to repurpose a *DM42* you own already, so you may save costs. This is explained in next chapter. Note a *DM42n* will not do it here.

The chapter thereafter (on p. 246f) shows how to update your existing *WP43*, be it bought or flashed, when a new firmware becomes available.

How to Create Your *WP43* by Flashing a *DM42*

First, save your data stored on your *DM42*, if applicable. You can do this in the following (reversible) procedure which you will need anyway:

1. Start your *PC*. Take your *DM42* and turn it on; then press

   SETUP

   then either

   1 **File >**

   or

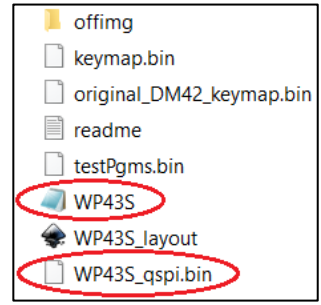
   5 **System >**
   2 **Enter System Menu**
   4 **Reset to *DMCP* menu**

2. Now connect your *PC* to the calculator *Micro-USB* (*B*) socket using a suitable data cable. Ensure it connects properly also on the calculator side – cutting back the plastic isolation a bit may be necessary. Then press…

   ☐ **Activate *USB disk***. The *FD* of your *DM42* should show up as an external mass storage volume on your *PC*.

3. You can copy the entire volume with your data into any suitable directory on your *PC* now.

4. Start the internet browser on your *PC* and fetch the firmware you find under *https://gitlab.com/rpncalculators/wp43/-/releases*. You will be asked to open a zipped folder. Extract it to any suitable location on your *PC*. Open its root directory and you will find this →



DRAFT

5. Copy `WP43.pgm` and `WP43_qspi.bin` onto the *FD*.[141]

   Then logically disconnect the *FD* via [EXIT] [ENTER↑]. Leave the cable connected since flashing will draw some current, so the little coin cell in the calculator will appreciate your support.

6. Press [4] **Load *QSPI* from *FAT***. When done, press any key (causing a reset), then

   [EXIT] and

   [3] ***DMCP* menu**.

7. Press [SETUP]

   [5] [2] [4]

   [3] **Load Program**. Scroll down to **`WP43.pgm`** and press

   [ENTER↑] [ENTER↑]. Wait ~ 15 s for flashing completed. Then press

   [EXIT] [EXIT] [1] and the bottom left key. Thereafter, your *WP43* is up and waiting for your commands. You may disconnect the cable now.

Finally, turn to p. B-32, cut out the print there, span it over the keyboard, and operate the keys through it.

Alternatively, you can also download `WP43_layout.svg` from the *Gitlab* repository, print it to scale, cut, and apply this overlay (cf. also pp. 184f) – the picture here shows the 17 keys needing stickers then.

Should you ever want to leave *WP43*, enter [MODE] DMCP for returning to the *DMCP* system. If you had chosen

[141] We recommend deleting any other files `*_qspi.bin` and `*.pgm` from the *FD* since the calculator may attempt loading these instead of the two correct files.

Option 1: Copy `keymap.bin` onto the *FD* unless you have done before. This will reassign the keys to match the *WP43* layout also when leaving *WP43*.

Option 2: Copy `testPgms.bin` onto the *FD*; it will give you some test programs pre-entered.

option 1 in f. 141, the key assignments will stay as they were in *WP43*; else the EXIT/ON functionality will fall back to the bottom left key.

To retrieve the original *DM42* keyboard layout (as pictured on p. 179), see https://technical.swissmicros.com/dmcp/devel/dmcp_devel_manual/#_recovery_to_required_keymap. To return to the previous *DM42* firmware, look up https://technical.swissmicros.com/dm42/doc/dm42_user_manual.html#quick_update_guide – just activate the *FD* by pressing USB first and continue with step 2 there then.

How to Update Your *WP43*

If you have *Free42* still running on your *DM42* then proceed as explained in previous chapter. But if *WP43* is installed on your calculator already then you can update as follows:

1. Start your *PC*. Turn your *WP43* on.

   SAVE **your programs and data!** Then leave *WP43* by pressing

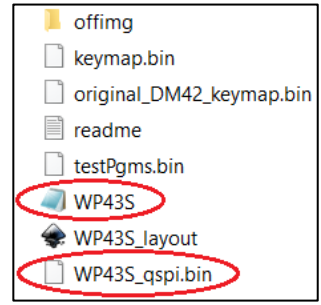
   MODE DMCP to return to the *DMCP* menu. If you had copied `keymap.bin` of *WP43* before last flashing, key assignments will stay as they were in your *WP43* when navigating in the *DMCP* system – else the *DM42* assignments will become valid (cf. p. 179).

2. Connect your *PC* to the calculator *Micro-USB (B)* socket using a suitable data cable. Ensure it connects properly also on the calculator side – cutting back the plastic isolation a bit may be necessary. Then press…

   6 **Activate *USB disk***. The *WP43 FD* should show up as an external mass storage volume on your *PC* now (if it does not then either your cable is no data cable or it does not connect or is otherwise defective).

3. Start the internet browser on your *PC*, go to https://gitlab.com/rpncalculators/wp43/-/releases and fetch the latest 'Firmware for DMCP' you find there. A zipped folder will be downloaded. Extract it to a suitable location on your *PC*.

   You will find this in its root directory →



4. Copy `WP43_qspi.bin` and `WP43.pgm` (optionally `testPgms.bin`, too) onto the

DRAFT

WP43 FD.[142]

Then logically disconnect the *FD* via

[EXIT] [ENTER↑]. Leave the cable connected, however, since flashing will draw some current, so the little coin cell will appreciate your support.

5. Press

   [4] **Load *QSPI* from *FAT***. When done, press any key (causing a calculator reset), then [EXIT].

6. Press any key to return to the *DMCP* menu. Then press

   [3] **Load Program**. Scroll down to **`WP43.pgm`** and press

   [ENTER↑] [ENTER↑]. Wait some 15 s for flashing completed. When done, press any key (causing another calculator reset), then [EXIT], and your updated *WP43* will be booted reset. Feel free to disconnect the cable now.

7. Now you can recall all your programs and data you have saved in step 1 above:

   [I/O] **LoadAll** or recall just a subset (cf. LOAD… in the *IOI*).[142]

   Then resume your work on your updated *WP43*.

Sometimes (seldom), an update of the *DMCP* on your *WP43* may be required. See https://technical.swissmicros.com/dm42/doc/dm42_user_manual/#DMCP_update_guide for how to do this.

[142] If you opted for a new set of `testPgms.bin`, these programs will be loaded when your *WP43* is booted (reset). Note, however, that LOADALL will load the previously saved old set of test programs again, overwriting the new ones.

Thus, if you want to get *your own* programs back after a firmware update and need a new set of test programs for any reason whatsoever, you shall first write *your own* programs onto the *FD* individually using WRITEP – in addition to SAVE (cf. step 1). Then update your *WP43*.

Eventually, when you come to step 7 above, do not call LOADALL but LOADV, LOADR, LOADΣ, LOADEQ, and LOADSS instead; then recall *your own* saved programs by multiple calls of READP. This way you will get back everything you created and have the generic test programs updated.

DRAFT

APPENDIX G: TROUBLESHOOTING GUIDE

First a Few Words About Coin Cells

Your *WP43* is powered merely by a single *CR2032* coin cell like other *SM* calculators. This allows for a low hardware profile but carries some disadvantages elsewhere. In particular, there are lots of cheap coin cells on the market well capable to deliver an open-circuit voltage of $>3\ \mathrm{V}$, but quickly render unusable under load. Powered by such a cell, your calculator may show various less than funny properties (it may even look completely dead!) which can be retraced to voltage drops under load. Especially when running programs or advanced functions, your *WP43* draws some *milliamps* from its battery – a *WP43* looking innocent before may, e.g., reset suddenly when its battery is close to depletion.

Also low temperatures let the voltage of such cells break down.

Thus, if you observe anything weird on your *WP43* and it is cold, turn it off, put it in a room of about 20°C or more and let it warm up to thermal equilibrium. Then save (SAVE) your programs and data if this is possible still.

Take a big breath. Now, if you suspect the battery being the root cause of your troubles (it may well be), turn your *WP43* off, open it, and remove its battery (see *https://technical.swissmicros.com/dm32/doc/dm32_user_manual.html#_battery*). Look at both sides of it: if the producer put a small sticker on one pole (usually on +) in firm belief that would keep toddlers from swallowing it (it doesn't!), remove it.

You should measure its voltage now – but most digital voltmeters (with an input resistance of some $\mathrm{M}\Omega$) will tell you its open-circuit voltage only: even an almost depleted battery may still show ~3 V there. You are far better off as long as you find a moving-coil voltmeter somewhere: due to its lower input resistance of some $\mathrm{k}\Omega/\mathrm{V}$ it will give you better information about the real state of your cell. Any reading significantly below 3.0 V shall lead to battery exchange. Insert a fresh quality cell[143] and verify if your observations persist. **Be sure you did this before reporting any troubles.**

Furthermore, note that such a coin cell has significantly less capacity than, e.g.,

[143] Renowned brands control their cell production in a way you can trust them – but there are many fakes of such cells, too. Personally, I gathered good experiences with *Amann*, *Duracell*, *Energizer*, and *Varta*. *Panasonic* and *Toshiba* are reported being good, too.

Generally speaking, the internal resistance of *CR2032* lithium cells is significantly higher (~ 10 Ω even for good cells) than in other common battery chemistry systems (~ 0.3 Ω for alkaline *AAA* batteries). If *SM* had chosen their top calculators to run on two *AAA* batteries instead, they had become a bit thicker at their top but had saved us a lot of troubles.

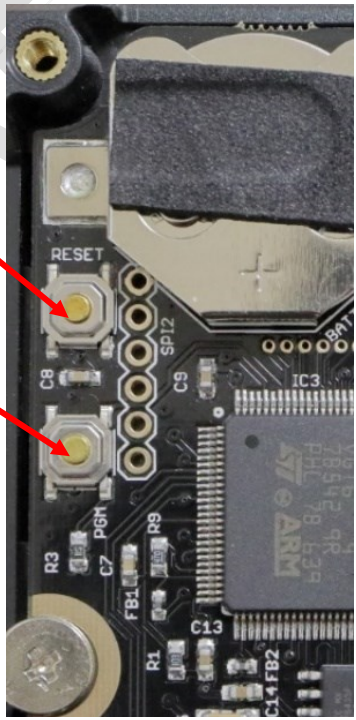
AAA batteries for obvious reasons. Looking at the differences in power consumption stated on p. 178, you will realize that using your *WP43* mainly for advanced math will deplete the cell 40 times faster than with occasional number crunching – it will last for many months with the latter.

Calculator Freeze

There are several ways to put your calculator in a freeze state wherein the *LCD* may show anything (or nothing at all) and your calculator will not react on any key you press. Usually, actuating the RESET button through the hole on its rear side should bring it back to life.

If this did not work, however, power your *DM42* via *USB*: this may suffice to reset it; if it does and the calculator was at room temperature, then the battery is the culprit.[144] After exchanging it you are usually done; else do:

1. open your calculator by unfastening the two Phillips bolts at the top of its backside. You will find a *PCB* with its top left looking like this (cf. p. 182 for an earlier *PCB* and p. 183 for one of a *DM42n*). In any case, you will notice two small buttons, labeled RESET and PGM (or BOOT0).



2. Now:
   a. Press and hold PGM.
   b. Press RESET for ~ 2 s.
   c. Release PGM. This shall reset your *DM42* and put it in bootloader mode.

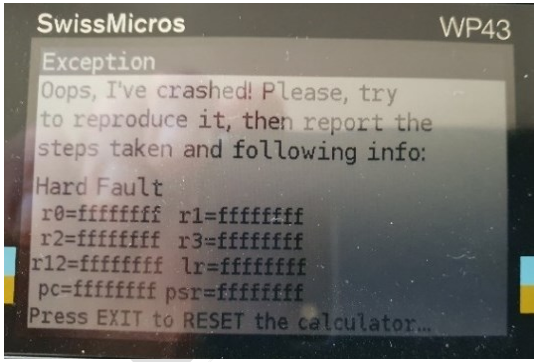
      If it does then you can flash your calculator now as explained on pp. 244f, if necessary.
   d. If it does not, however, and especially if the *LCD* stays blank also under *USB* supply, this may point to a real hardware

[144] Cf. previous chapter. Almost depleted cells may be unable to process a power on, a reset, or even a single keystroke, leaving you wondering what has happened to your calculator (which tells you nothing).

DRAFT

problem. Read also the other cases in this *App*. If nothing applies, ask on the *SM* forum. If this does not help either, contacting *SM* may become inevitable.

Exception



When you see a screen like this, you have experienced a crash, presumably due to a firmware bug. Please act as requested. If the crash is reproducible, record its boundary conditions and report this incident to us, please. We will investigate it and return to you in this matter as soon as possible, either with advice or with a new firmware release.

Fresh Quality Battery Constantly Low

This was observed with a factory-fresh calculator in 2020: The low-battery indicator was lit and the command BATT? persistently returned 1.21 V, although the battery was actually good and measured over 3 V (open-circuit). If resetting the calculator, removing the battery, updating the firmware, etc. does not change the output of BATT?, then inspect the *PCB*. There may be an improper soldering next to FB1 (a.k.a. a solder bridge – compare the picture on previous page).

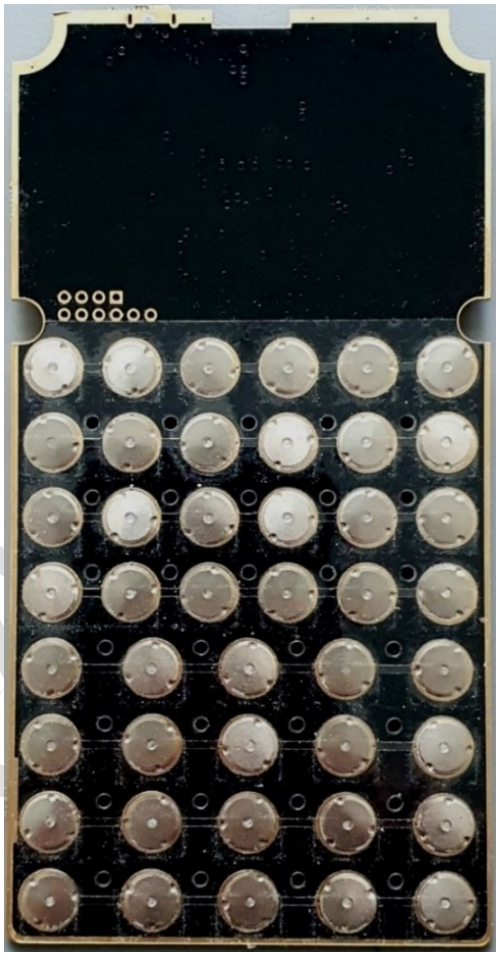
If you have proper tools and experience, feel free to fix this yourself – else contact *SM*.

Individual Keys not Reacting Properly

This may be caused by switching domes faulty or misplaced. Lesser faults may cause particular keys requiring more force to actuate or just “feeling different”.

To access the keyboard side of the *PCB* carrying the switching domes, open your calculator by unfastening the two bolts at the top of its backside like you do for changing the battery. Thereafter carefully release the *LCD* connection Ⓐ,[145] then unfasten the two wide-headed Phillips bolts Ⓑ. Note these operations are at your own risk.

The picture here shows an early *PCB* of 2017, but positioning accuracy of the 44 domes in production could still be improved six years later. Any anomaly you detect on this side of your *PCB* could cause a malfunction of the corresponding key.[146]



Keymap Trouble

If you had loaded a `keymap.bin` not matching the layout you want or you do not find the keys properly assigned (and you are not in user mode!) then proceed this way (presuming both calculator and *PC* are running and connected properly):

1. Find where the *menu* MODE went on your calculator keyboard and open it.

[145] *https://technical.swissmicros.com/dm32/doc/dm32_user_manual.html#_disassembly* contains some drawings in this matter.

[146] If you are handy than you can try, at least, a cure available for little money at *https://www.swissmicros.com/product/dm42-replacement-foil-with-domes*.

2. Call [▭] [DMCP] to return to *SM*'s *DMCP* menu.
3. Press [6] **Activate *USB disk***.
4. Start the internet browser on your *PC* and look up *https://technical.swissmicros.com/dmcp/devel/dmcp_devel_manual/#_recovery_to_required_keymap* . Act accordingly.
5. Press [EXIT], then the bottom left key – else you will run into the same trouble again.
6. Go to p. 244 for proper flashing.

Other Trouble

Of course, there is a plenty of opportunities more to kill or brick your calculator. Drop it on rocky ground, sink it in a puddle, leave it in the laundry for too long, spill coffee over it, break its display, leave a leaking battery in it, etc. etc. – just your imagination is the limit. All these mishaps,[147] however, don't affect the firmware in any way but the basic hardware. Thus, please check *https://technical.swissmicros.com/dm42/doc/dm42_user_manual.html#DMCP_update_guide* and the chapters following there for advice. If nothing helped, turn to *SM*.

DRAFT

[147] *TN*: I came across this cute little term in Indian newspapers. It used to appear there as *kitchen mishap*, frequently linked to poor elderly married women dying because of exploding gas cylinders or the like. In my opinion, *mishap* sounded far nicer than accident or worse.

APPENDIX H: HISTORIC REMARKS AND CAVEATS FOR YOUNGER USERS (WHAT GRANDPA STILL KNEW)

In company-internal communication, *HP*'s pocket calculators carried individual names beyond their type numbers. And each product line carried a family name:

1. ***Classics*** for the *HP-35* of 1972 and its immediate successors *-80*, *-45*, *-65*, *-70*, and *-55*
2. ***Woodstocks*** for the smaller devices following, i.e. the *HP-21, -25, -22, -27, -25C,* (*-67*,) and *-29C* (*-19C*)
3. ***Spices*** for the *HP-31E, -33E/C, -38E/C, -32E*, *-34C*, and *-37E*
4. ***Nuts*** for the *HP-41C* and its siblings *CV* and *CX*
5. ***Voyagers*** for the compact landscape calculators *HP-11C, -12C, -15C, -16C,* and *-10C*
6. ***Pioneers*** for the line of portrait calculators following (encompassing also algebraic models), i.e. the *HP-17B*, *-27S*, *-22S*, *-32S*, *-42S*, *-14B*, *-20S*, *-21S*, to the *HP-32SII* whose production ceased in 2001.

This was the last real product family of pocket calculators; single calculator designs followed. The *Classics*, *Woodstocks* and *Voyagers* had a particular fame for their mechanical ruggedness – various stories are told. You can still find many of them working without any flaw until today, for more than 50 years.

Classics, *Woodstocks*, and *Spices* featured power-hungry *LEDs*, so they enclosed rechargeable battery packs (usually NiCd – lithium-ion batteries became available many years later). Though the latter two families also contained a serious design flaw in their charging circuitry, actually killing thousands of them, each in a fraction of a *second*:

> **WARNING: Never (!!!) connect any *Woodstock* or *Spice* to the mains** unless you are sure it contains a good battery! And since you can actually never be 100% sure about a rechargeable battery, just never connect any of them to the mains – charge the batteries outside of these calculators!

Up to and including the original *Voyagers*, *HP's* calculators were built to be repaired. You can disassemble them easily, more or less. Getting suitable electronic spare parts after decades may become a challenge, however.

DRAFT

Checking the forum in the *Museum of HP Calculators* (*MoHPC* at *https://www.hpmuseum.org*) will often help. On the other hand, disassembling a recent *Voyager* (e.g., a *HP-12C Platinum* or *HP-15C CE*) is a challenge – you may open the case (although they use tiny bolts with quite soft heads) but the *PCB* is fixed on the keyboard by heat stakes which can be opened only once.

Keys and Keyboards

HP had a reputation for producing outstanding, solid pocket calculators also hardware-wise. This was earned with the *Classics* and kept until today, although most factors of distinction were sacrificed to cost reduction as time went on. Remember the *HP-35* was built by engineers for engineers and a major investment when it was launched – 395 US$ of 1972 correspond to 3022 US$ of today. And beyond the *USA*, these calculators were even more expensive (the *HP-35* selling for more than 1000 DM in Germany in 1974 corresponding to 1823 € of today), rendering them unobtainable for most university students until prices dropped in the late Seventies.

What made *HP's* calculators special in the beginning? They featured …

1. double-shot keys – their legends withstood all wear and tear,
2. individual keys – each key sat in its own little niche and moved entirely independent of its neighbors,
3. a precise click which indicated reliably that the key had registered,
4. and they were built to last (most times – cf. previous chapter).

Double-shot keys were abandoned while the *Pioneers* were made.

The last calculators featuring individual keys were the *Voyagers* – and since the *HP-12C* is produced continuously from 1981 until today, you can still buy such a calculator new (though accessing its keys is a challenge, cf. above). Actually, however, todays way of making the keys all together in one shot (cf. *App. A*) is simply good enough – individual keys don't pay anymore.

And feature 3 was actually achieved by each button acting on a metallic snap contact. For its first pocket calculators, these contacts were gold-plated and designed by *HP* itself. Later, circular switching domes became commercially available. Nowadays, *HP's* famous 'rotate-and-click' keyboards (actually showing very little rotation) result from a careful combination of hardware decisions and software parameter setting, a combination *HP* tuned very well.

DRAFT

Though also this expertise fades as *HP's* former calculator division decreased and is actually almost extinct now.

Reverse Polish Notation

RPN is traced to *prefix* and *postfix* mathematical systems developed by *Jan Łukasiewicz* (pronounced ~*Wookus-jéwidch*) in the 1920's, and proposed in his book of 1951 (cf. *OM*, f. 1). *Prefix notation* means placing operators before the numbers they operate on instead of between them.

> 17 + 4, for **example**, reads + 17 4 with *prefix notation* (called *Polish Notation* in honor of *Jan*). In *postfix notation* (hence *Reverse Polish Notation*), it reads 17 4 + instead.

Note, however, that many mechanical calculators (a.k.a. *adding machines*) worked alike before *Łukasiewicz* already.[148] Entering and storing two numbers in some kind of *registers* and then performing an operation on them is a natural implementation.

In the years that followed [*Jan's* book], computer scientists realized that *RPN* or *postfix notation* was very efficient for computer math. As a postfix expression is scanned from left to right, operands are simply placed into a last-in, first-out (LIFO) *stack* and operators may be immediately applied to the operands at the bottom of the *stack*. By contrast, expressions with parentheses and precedence (infix notation) require that operators be delayed until some later point. Thus, the compilers on most modern computers converted statements to *RPN* for execution. (In fact, some computer manufacturers designed their computers around *postfix notation*.)

At the time that the *HP-35* was introduced, other pocket calculators typically used a partial algebraic model. That meant they could evaluate trivial expressions like 4+5 but couldn't handle anything that involved parentheses or algebraic precedence. The technology of the time didn't allow for full algebraic compilers in pocket calculators.

RPN allowed *HP* to produce a pocket calculator that could evaluate [almost] arbitrary expressions using the available technology. For many, learning a new style of entry was a small price to pay to be able to evaluate arbitrary expressions

[148] The following text is quoted from *Dave Hicks'* online *Museum of Hewlett-Packard Calculators* (as found on 2023-11-11), but with some additions in brackets.

on a calculator. Once the technology to produce algebraic compilers could fit into a pocket calculator, most *RPN* users had decided that *RPN* was more efficient and consistent for the user as well as for the calculator. Also, because subexpressions are evaluated as they are entered, entry errors are more obvious with *RPN*. On an algebraic calculator, omitting an opening parenthesis, may not lead to a calculation error until much later when an entire subexpression is evaluated.

Another advantage to *RPN* is consistency between machines. Early algebraic models had differing limits of the complexity of the expressions they could evaluate. For example, *TI* [*Texas Instruments*] catalogs from the late 70's listed how many levels of parentheses and pending operations each model could handle. Even today if you begin to use an algebraic calculator, you need to determine just "how algebraic" it really is. For example, some switch to *RPN* for unary [a.k.a. *monadic*] operators (i.e. 5 SIN rather than SIN(5) or even SIN 5 =), some are still missing parentheses and/or precedence and many can't directly enter an expression like:

$$4 + 5/6 + 7$$

even though they claim to allow expressions to be entered as they are written.

Why Does ENTER↑ Work the Way It Does? And Why Are there Automatic Stack Lifts and Drops?

HP's very first electronic calculator, the desktop model *9100A* of 1968, featured only a *3-level stack* – but the contents of all three levels were always displayed on the screen of a small cathode ray tube: the *keyboard (entry) register* **X**, the *accumulator* **Y**, and the *temporary register* **Z**.

Earlier mechanical calculators worked with a *keyboard*

DRAFT

register and an *accumulator* only. The third *register* became necessary to enable the calculator to deal with slightly more advanced mathematical expressions.

See the center keyboard section of a *HP-9100A* pictured above. Solving a little problem like $123 / (45 + 67)$, beginning with a clear *stack*, the following steps had to be executed leading to consecutive displays as shown in separate columns below:

z	0	0	0	123	123	123	123	123
y	0	123	123	45	45	112	123	1.0982...
x	123	123	45	45	67	67	112	112
Input	**123**	↑ [149]	**45**	↑	**67**	+	↓ [150]	÷

But a pocket calculator like the *HP-35* could display only one *register* with its *LED's* for space and power reasons – and that *register* had to be **X** to give feedback during keyboard entry.

In consequence, to show the calculation result supplied in the *accumulator* **Y**, *y* must be moved into **X**. Thus both + and ↓ have to be executed immediately after each other, preferably in a single new operation + (with an *automatic stack drop*, cf. *OMS 1*). The same applies to the other *dyadic* operations as well. On an *HP-35*, it looks like this:

z	0	0	0	123	123	0	0
y	0	123	123	45	45	123	0
x	123.	123.	45.	45.	67.	112.	1.0982...
Input	**123**	ENTER↑	**45**	ENTER↑	**67**	+	÷

On the other hand, such an intermediate result in **X** has to be shifted out of the way into the *accumulator* **Y** again before entering the next number into **X**. Thus,

[149] This key worked like ENTER↑ did on subsequent calculators.

[150] In fact, ↓ did exactly what DROP↓ does on your *WP43*; this desktop calculator key disappeared on *HP*'s pocket calculators for space reasons (see an *HP-35* of 1972 overleaf). ROLL↓ survived as R↓ – it does almost the same. ROLL↑ equals R↑ .

[↑] is executed automatically then (= *automatic stack lift*, cf. *OMS 1*). [151]

But this is not required, of course, if [↑] has just been pressed before. This is why *ASL* is disabled after [ENTER↑].

ASL is also disabled after CLX since the new number entered shall simply replace the one just deleted in **X** – an extra zero on the *stack* makes no sense.

Σ+ and Σ− are dedicated commands for adding (accumulating) and subtracting statistical, measured, or counted data (see *OMS 2*). They were designed just for data input since their first appearance on the *HP-80* and *HP-45* in 1973 and were not really meant to be used in *stack* calculations.

This applies in analogy also to the six *Matrix Editor* commands mentioned in *Section 1*.

[151] *Stack register* **T** was just added to **X**, **Y**, and **Z** to provide one temporary *register* more, allowing for more complex calculations under the boundary conditions of the early 70's. Actually, that purpose is reflected in its name, maybe also that it is the top *stack register*.

Likewise, we added **A**, **B**, **C**, and **D** to the *WP 34S* since dealing with *complex numbers* on it required two *registers* for each such number (similar to the *HP-15C*). Our resulting *8-level stack* turned out being also advantageous for simpler objects (field proven since 2011, see also *OMS 1*).

How Many Stack Levels Are Necessary?

Millions of *HP RPN* pocket calculator users were (and are) happy just working with a *4-level stack* plus at least one extra *GP register* since 1972. *Heathkit* offered an *RPN* pocket calculator for airplane navigation featuring a *5-level stack* a bit later. There was found, however, no calculator hardware with larger *stacks* until *HP* launched its *HP-28C* featuring *RPL* and an 'infinite' *stack* in 1987. Although *HP's RPL* calculators significantly exceeded shirt pocket size immediately, the *RPL* line lived until production of the *HP-50g* ceased in 2015. *RPN* pocket calculators continue being made until today.

> Obviously you can't have *top level repetition* (cf. *OMS 1*) with an infinite *stack*, and also [R↓], [R↑], and [FILL] make no sense there for the same reason. All in all, *stack* handling in *RPL* requires many more commands (DUP, SWAP, DROP, OVER, ROT, UNROT, ROLL, ROLLD, PICK, UNPICK, PICK3, DEPTH, DUP2, DUPN, DROP2, DROPN, DUPDUP, NIP, and NDUPN) than just [ENTER↑], [x⇄y], and [R↓] in common *RPN*. While *RPL* certainly has its merits, many potential users (advanced students and professionals) prefer the easier approach of *RPN* and, last not least, a true pocket calculator.

The size of the traditional *RPN stack* was certainly a compromise between the technical resources and engineering cost in 1972. We are not bound to that anymore. Decades of field experience solving real-life problems have shown that four *stack* levels suffice in almost all cases if you calculate inside out as recommended. Though *stack* overflow may happen silently without a warning and the odds are high you will not notice it immediately – but the consequences for your calculations may be catastrophic. With an *8-level stack*, however, you are definitely on the safe side as demonstrated comprehensibly in *OMS 1*.[152] **Calculating *sans soucis*!™**

DRAFT

[152] Find a historic discussion about this topic on *Valentin Albillo's* excellent *HP* website *https://albillo.hpcalc.org/threads/HP%20Thread%20VA100b%20-%202005-07-15%20-%20How%20many%20stack%20levels.pdf* .

Side note:

Obviously, *HP* had thought about a dedicated statistical calculator in the early 70's (past *HP-80* – see the key holes), but that project never really materialized. Just this prototype case was discovered and published by *aj04062* (*USA*) at the *MoHPC* in 2024.

The four lowest rows of keys are all set with Σ+ bottom right. Remaining keys can be guessed:

1. 1/x, LN, LOG, x^2, and Q,
2. $(\chi^2)^{-1}$, t^{-1}, $\bar{x}$, s, and ■,
3. F^{-1} in the center of 3rd row,
4. eventually ENTER↑, CHS, maybe CLΣ, and CLx.

On either side of F^{-1}, both pairs STO and RCL as well as x⇄y and R↓ were located most probably. FMT denotes a modified *t-test*.

Outputs of LIN and QUAD REGRESSION were interesting, since r^2, a_0, a_1, and a_2 (if applicable) should have been returned. So QUAD had overwritten the entire *4-level stack* then. And quadratic fitting required at least nine (*GP*) *registers* for accumulating the statistical sums n, Σx, Σx^2, $\Sigma x^2 y$, Σx^3, Σx^4, Σxy, Σy, and Σy^2 (see p. 290).

For sake of layout coherence, Q^{-1} and Q should be swapped though.

How Many Stack Levels Shall Be Displayed?

Millions of *RPN* pocket calculators built so far displayed x only – y, z, and t are quietly acting unseen. Users were doing all sorts of tricks using these four *registers* – just tracking y, z, and t in their minds. Silicon memory was scarce and expensive – human memory was cheaper for decades.

Even *HP's RPL* calculators, launched between 1987 and 2015, with their 'infinite' *stacks* displayed no more than their bottom four *stack registers*, although many of the tricks making sense on a *4-level stack* became complex or obsolete there.

With an *8-level stack* and *RPN*, nobody expects you tracking y, z, t, a, b, c, and d in your mind – it is simply not necessary at all. You know how the *stack* contents move, and they will show up automatically when needed (cf. *OMS 1*).

No mathematical function on your *WP43* operates on more than x, y, z and t at once. Thus, and assuming human mental abilities did not deteriorate in last decades, displaying more than four *stack* levels carries no lasting benefit. This holds in particular since the odds for *stack overflow* in real-world calculations are reduced to zero (!) when employing an *8-level stack*, calculating inside out as recommended (cf. examples in *OMS 1*).

For the same reason, we omit heading indicators X, Y, Z, and T on display. You chose a *WP43* for yourself, so you are certainly able to remember these four letters naming the bottom four *stack registers*.

It is even debatable if more than the bottom one or two *stack* levels need to be displayed at all. This decision is up to you – the command DSTACK will help.

Precision and Accuracy of Calculations with Real Numbers

Before the *HP-35*, precision of most technical calculations was limited to 3 - 4 significant digits for decades – that was what you could read on a quality slide rule operating carefully. Note this did suffice for bringing so-called 'astronauts' to the Moon and sending two *Voyagers* to the outer planets of our solar system and into interstellar space beyond, as we all know.

Then the *HP-35* and its successors changed personal computing completely – featuring a precision of 10 digits or more. In wise self-restraint, however, startup default display format of many *HP* pocket calculators for science and engineering became FIX 4.

Since then, paradigms changed again: Now, precision of your *WP43* and its

DRAFT

siblings goes far beyond what is necessary for most applications even in science. As mentioned above, internal computations are usually carried out with 39 digits.[153] Displayed results (rounded to 34 digits) should show a relative error < $\pm 1\times10^{-33}$ per plain operation. See some random **examples** in *SDC*:

- ***n*** [1/x] **2*n*** [×] **2** [−] returns precisely 0. for almost all primes ***n*** < 700 (except **7** [1/x] **14** [×] **2** [−] **2** [/] returning 5×10^{-34}, like 641; for 67, 83, and 683, results are -5×10^{-34});
- **7** [sin] [arcsin] **7** [−] **7** [/] returns 1.4×10^{-34};
- **7** [ln] [e^x] **7** [−] **7** [/] = 2.9×10^{-34} while **7** [e^x] [ln] **7** [−] returns plain 0.;
- **7** [√x] [x^2] **7** [−] **7** [/] = -2.9×10^{-34} while **7** [x^2] [√x] **7** [−] returns plain 0;
- **7** [ENTER↑] [$\sqrt[x]{y}$] **7** [y^x] **7** [−] **7** [/] returns -4.3×10^{-34}.

For comparison, SLVQ shows an inaccuracy of $\pm2\times10^{-33}$ (cf. *OMS 4*) while nested advanced operations like SOLVE(∫) may be inaccurate up to some $\pm6\times10^{-33}$ or even more due to many elementary operations executed therein, whose inaccuracies accumulate.

There is a quasi-standard established to find out about processors, firmware, precision, and 'accuracy' of calculators – compute $arcsin\{arccos[arctan(tan\{cos[sin(9°)]\})]\}$ (although this formula is mathematical nonsense). An ideal brainless number cruncher of infinite internal precision will return exactly 9° or equivalent (see, e.g., https://www.wolframalpha.com/input/?i=arcsin(arccos(arctan(tan(cos(sin(π/20))))))).

Real calculators (computing with a finite number of digits) deviate for obvious reasons. Your *WP43* returns

8.999 999 999 999 999 999 999 999 999 937 535 = $9 - 6.246\,5\cdot10^{-29}$.

If you are interested how other calculators have performed in this test, look at http://www.rskey.org/~mwsebastian/miscprj/results.htm. But be aware that some calculators may cheat or lie to you based on their guesses what the expected results shall be.[154]

[153] There may be even more (see *Background*… below). Feel free to use the expressions $\left(729^{\,33.5}/_{3.201}\right)-1$ and $\left(531\,441^{\,1072.5}/_{9.6435}\right)-1$ for testing any calculator you may get your hands on. Your *WP43* returns precisely **0.** for both.

DRAFT

Far simpler is another test, but very potent: Enter **10 000 001.** and press [x^2] just seven times. Your *WP43* will return

$$1.000\,012\,800\,081\,280\,341\,377\,066\,802\,645\,679\times10^{896}.$$

This is the most precise *real* result known for any pocket calculator.[155] Nevertheless, only its first 32 digits are correct! For comparison, enter **10 000 001** (without the decimal point) and press [x^2] seven times again. Your *WP43* will return

$$100\,001\,280\,008\,128\,034\,137\,706\,680\,264\,566\,942\times10^{861}$$

instead, which actually is an 897-digits *long integer*, i.e. precisely

100 001 280 008 128 034 137 706 680 264 566 942 362 065 259
381 704 558 672 471 663 975 250 935 637 066 049 668 878 686 323 398 085
866 658 787 740 848 661 692 888 859 950 737 447 654 753 345 168 002 963
740 073 746 995 710 556 007 241 670 099 300 929 440 107 854 408 072 554
911 252 317 789 928 499 808 013 093 965 927 574 941 133 244 381 547 633
598 449 089 802 869 082 118 094 468 776 017 826 651 186 922 820 900 475
103 999 094 028 004 541 163 740 697 539 733 888 428 367 675 593 254 423
047 914 478 918 135 752 102 429 288 143 361 653 459 171 040 814 216 776
401 746 346 343 970 859 728 390 095 407 066 078 152 415 625 359 663 834
513 241 403 709 468 936 478 003 065 366 991 757 932 039 263 495 546 002
578 420 990 239 188 092 834 549 068 962 510 444 861 858 929 119 375 002
542 434 572 320 889 769 109 362 436 235 325 985 158 624 722 403 373 962
502 148 033 363 010 732 930 713 924 088 117 757 151 012 106 967 826 289
242 967 898 516 694 703 304 135 759 892 818 613 724 712 834 463 748 230
478 358 549 260 032 564 540 002 953 869 332 914 454 490 582 230 113 562
493 732 890 458 571 501 560 870 702 174 970 266 185 257 957 423 611 226
456 640 106 680 000 341 376 000 812 800 001 280 000 001

DRAFT

[154] This behaviour has a parallel in airliners: so far (2024), Boeing's airplanes execute the pilot's commands directly while Airbus' execute what its software 'thinks' the pilot should have done under the conditions it observes. The latter may turn out dangerous and actually caused at least two fatal crashes.

Although a calculator can't crash this way, there are risks still. Actually, the problem with a calculator lying to the user is not the lie itself but that you never know if it is lying or telling you the truth – the calculator becomes a suspicious device, and trust in its reliability is destroyed reliably (parallels in recent *US* administrations are obvious – also some other societies are competing 41 years after 1984, and just 77 years after *George Orwell* wrote his vatic novel).

[155] If we were marketeers we would praise *WP43* as *the most precise calculator on the worldwide market – second to none*. Since we aren't we tell you that the *WP 34S* with DBLON and *Free42* are equally precise.

Taking this test further will increase inaccuracy – after 27 presses of [x^2] only the first 25 digits of the returned *real* result will be correct.

Such outputs qualify the concept of '*exactness'* or '*accuracy'* as a whole – they may deviate from your expectations but are the best results nevertheless you can get from a calculator of today with finite precision, not lying to the user.

At the bottom line, your *WP43* will return 34 digits always but not all of them are guaranteed being true.

Furthermore, there are inaccuracies caused by the limited precision of your input – these are usually far greater. Note the *least accurate constituent* (*LAC*) of your computation will determine the resulting accuracy – your result can never have more significant digits than your *LAC* (see pp. 298ff for a more comprehensive treatment of this matter). Watch it!

And do not just believe in a result displayed – do some quick order-of-magnitude estimates yourself, check and verify! A critical mind will help always.[156]

About the Classic *Mach Number* Formula

Let's return to said vintage formula (cf. *OMS 1*):

$$\sqrt{5\left(\left[\left\{\left(1+0.2\left[\frac{CAS}{661.5}\right]^2\right)^{3.5}-1\right\}\{1-6.875\times10^{-6}\times PA\}^{-5.2656}+1\right]^{0.286}-1\right)}$$

Nowadays, evaluating expressions like this is no real problem anymore also on many *algebraic* calculators. Here is its solution using a more modern *HP-12C Platinum* in *ALG* mode:[157]

[156] Before electronic calculators, the resulting order of magnitude had to be pre-estimated and digits were few. Now, digits are far more than enough but erroneous results in a totally wrong ballpark (due to input errors or operating mistakes) may easily slip through unnoticed.

(Parallels: Before the internet, information was scarce and had to be carefully searched. Nowadays, information is abundant and has to be carefully checked and verified, else blatant lies or propaganda – euphemistically called 'fake news' by interested parties – can and will spread easily. Entire less than democratic administrations and regimes may be (and were) erected where people were successfully confused by avalanches of 'fake news' so they couldn't tell fake from truth anymore. PRINCIPIIS OBSTA !)

[157] **ATTENTION:** This calculator allows for 13 open parentheses in *ALG* but knows nothing about operator hierarchy beyond – it calculates stubbornly from left to right.

DRAFT

350 / 661.5 = x^2 × .2 = + 1 = y^x[158] 3.5 = − 1 = STO 0
1 − (6.875 EEX CHS 6 × 25500) = y^x 5.2656 CHS =
× RCL 0 = + 1 = y^x .286 = − 1 = × 5 = $\sqrt{x}$

Note this algebraic solution (80 keystrokes) works inside-out as recommended for *RPN* in *OMS 1*, hence using = twelve times, though only one pair of parentheses and one *GP register* for solving. And x^2 as well as $\sqrt{x}$ are postfix operations also in *ALG* here, and CHS operates on the open input.

For solving this formula from left to right (almost) as written, even 92 keystrokes are required:

5 × ((((1 + (.2 × (350 / 661.5) x^2)) y^x 3.5 − 1) ×
(1 − (6.875 EEX CHS 6 × 25500)) y^x 5.2656 CHS
+ 1) y^x .286 − 1) $\sqrt{x}$

Please compare with the solution in *RPN* mode on the very same device:

350 ENTER↑ 661.5 / x^2 .2 × 1 + 3.5 y^x 1 −
6.875 EEX CHS 6 ENTER↑ 25500 × CHS 1 + 5.2656 CHS y^x ×
1 + .286 y^x 1 − 5 × $\sqrt{x}$

In *ALG*, you need 80 keystrokes at least compared to 63 in *RPN* here, i.e. 27% more on the same device![159]

For years, there are also ‘scientific’ calculators available featuring so-called ‘*natural textbook display*’ or ‘*MathPrint*’ or alike – actually those are targeted to the educational (mass) market[160] where problems typically culminate in inserting

---

The same is observed with the *HP-19BII*. Seems *HP* thought / thinks this ‘chain calculation’ paradigm suits business users. The lasting success of the *HP-12C* in these quarters proves it.

[158] This operation should rather read ^ in this mode.

[159] You may find more intelligent implementations of algebraic mode easily. A *CASIO fx-991DEX CLASSWIZ*, for instance, a calculator computing exclusively algebraically (with prefix $\sqrt{x}$), allows for entering the entire *Mach number* formula from left to right using 82 keystrokes – it can be even replayed and edited.

But beware: when this calculator turns off automatically after some minutes of inactivity, your nice formula will be gone without a trace.

[160] Mainly to schools, although some are called ‘high’.

values into formulas provided. Note that none of those calculators, however, can display the entire *Mach number* formula in a row.

Feel free to assess also the aptitude of those calculators for *explorational mathematics* (cf. *OMS 1*), coherence of their user interface, and consistency of their operation.

DRAFT

TN: Don't you ever confuse a *high school* with a German *Hochschule* – identical names though totally different ballparks! But since former *Fachhochschulen* (the minor siblings of universities and *Technischen Hochschulen*) were rebaptized in *universities of applied sciences*, the *Technischen Hochschulen* had to become *technical universities* to keep distance. Little substantial improvement – just an unlimited belief in pretty facades and an unbroken trend of euphemising lower quality. Marketing …

APPENDIX I: ADVANCED MATHEMATICAL FUNCTIONS AND TASKS

Lasciate ogni speranza, voi ch'e entrate!
(Dante Alighieri, Commedia Divina)

There's neither witchcraft nor sorcery in mathematics.
(unknown contemporary author)

Your *WP43* contains several operations covering advanced mathematics. Many are taken over from *WP 34S* and previous *HP* calculators, some are implemented here for the first time on an *RPN* calculator (cf. *App. D*). Find these functions described below in more detail than in the *IOI*, together with a few traditional pocket calculator functions matching the topic.

As explained in *OMS 1*, we presume you are able to read and comprehend mathematical formulas for *real* and *complex* domain functions. Ensure you understand the respective fundamental mathematical concepts, else leave these functions aside. By experience, it is only beneficial to use and apply something you overview and know the background of – else it may even become dangerous for you and your fellow men (¡no toca que desconoce!).[161]

Number Generating Functions

The following are all *monadic* functions except COMB and PERM.

Name	Remarks (cf. pp. 11ff for general information)
B_n , B_n^*	B_n returns the Bernoulli number for an *integer* ***n*** > 0 given in **X**: $B_n = (-1)^{n+1} \cdot n \cdot \zeta(1-n)$ B_n^* works with the old definition instead: $B_n^* = 2 \cdot \frac{(2n)!}{(2\pi)^{2n}} \cdot \zeta(2n)$ See p. 322 for $\zeta(x)$.

[161] Real world shows many sad examples where people full of good will caused large damages by applying tools they did not know sufficiently – or applying standard tools in areas where those were not applicable. There was, e.g., a major oil spill in the Gulf of Mexico caused this way. "*Wenn Dumme fleißig werden, wird's gefährlich*" (translated ~ "*It's becoming dangerous with fools becoming busy*"), a boss of mine used to say.

Name	Remarks (cf. pp. 11ff for general information)
COMB, PERM	For $y \geq x \geq 0$ and $x, y \in \mathbb{N}$, $$C_{y,x} = \binom{y}{x} = \frac{y!}{x!\ (y-x)!}$$ is the number of *combinations* and $$P_{y,x} = \frac{y!}{(y-x)!} = x!\ C_{y,x}$$ the number of *permutations* of x and y as explained in the *IOI* (cf. pp. 31 and 69, respectively). Note $C_{y,0} = 1$, $C_{y,1} = y$, $C_{y,2} = 0.5\ y\ (y-1)$. $C_{y,x}$ applies to the *binomial distribution* (see p. 269): In a *Galton box*[162] featuring y rows of pins and fed with 2^y balls, $C_{y,x}$ is the number of balls expected in column x of the box (start column counting with zero). $$P_{y,x} = \frac{\Gamma(y+1)}{\Gamma(y-x+1)} \quad \text{and} \quad C_{y,x} = \frac{\Gamma(y+1)}{\Gamma(x+1)\,\Gamma(y-x+1)}$$ are the general formulas working also for non-integer numbers and in *complex* domain.
FACTOR	… factorizes an integer input number. The output is a *real* matrix of two rows. Its 1st row contains the prime factors found and the 2nd row their exponents. E.g., **2 443 466 684 015 280** FACTOR returns $\begin{bmatrix} 2. & 3. & 5. & 7. & 11. & 17. & 123\,456\,791. \\ 4. & 3. & 1. & 2. & 1. & 1. & 1. \end{bmatrix}$, since it equals $2^4 \times 3^3 \times 5 \times 7^2 \times 11 \times 17 \times 123456791$. **ATTENTION:** If you suspect a number > 10^9 being prime, use PRIME? first – it is significantly faster than FACTOR in this matter. FACTOR operating on actual primes > 10^9 may take very long – see p. 325 and feel free to use EXIT to abort.

[162] *TN*: This is called «Planche de Galton» in French, "Galtonbrett" in German, and "macchina di Galton" in Italian, for example. Note the subtle differences in naming. Galton invented his box or board in 1889.

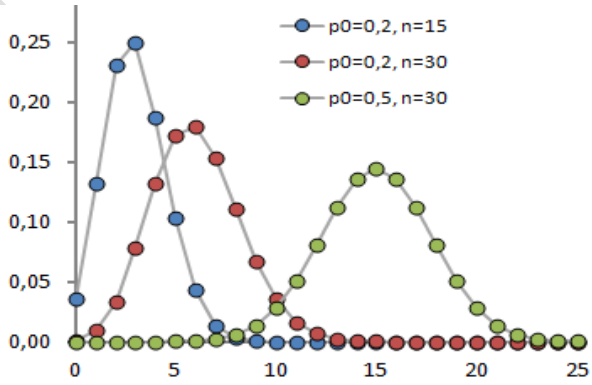
DRAFT

Name	Remarks (cf. pp. 11ff for general information)
FIB	For *integers*, FIB returns the *Fibonacci number* f_n with the counting number $n = x$. The *Fibonacci numbers* are defined as $f_0 = 0$, $f_1 = 1$, and $f_n = f_{n-1} + f_{n-2}$ for $n \geq 2$. With UNSIGN, f_{93} is the maximum before an overflow occurs. For *long integers*, f_{4791} is the maximum on your *WP43*. For non-integers, FIB returns the *extended Fibonacci number* $$F_x = \frac{1}{\sqrt{5}}\left[\Phi^x - \Phi^{-x}cos(x\,\pi)\right]$$ for any *real* or *complex number* $\boldsymbol{x}$, with $\Phi = {(1+\sqrt{5})}/{2}$ denoting the *golden ratio*.

Statistical Distribution Functions (*PMF*, *PDF*, *CDF*, etc.)

Stack-wise, the following functions stored in PROB are all *monadic*. Actually, they feature more parameters though, supplied in *registers* **I**, **J**, and **K** as applicable and mentioned below. In the following, the five discrete distributions are covered first, the nine continuous ones thereafter. Plots (taken from *Wikipedia*) show typical *PMF*'s or *PDF*'s.

Binom: ***Binomial*** *distribution* with the *number of successes* ***g*** in **X**, the *gross probability of a success* $\boldsymbol{p_0}$ in **I** and the *sample size* ***n*** in **J**.



BINOM_P returns

$$p_B(g; n;\, p_0) = \binom{n}{g}\, p_0^g\, (1-p_0)^{n-g} = C_{n,g}\, p_0^g\, (1-p_0)^{n-g}$$

(cf. COMB on p. 268 for explanation of the notation).

DRAFT

BINOM▲ returns $F_B(m;n;p_0)=\sum_{g=0}^{m} p_B(g;n;p_0)$ with the maximum number of successes ***m*** in $\mathbf{X}$.

The *binomial distribution* is fundamental for error statistics in industrial sampling, e.g., for designing sample test plans. Read more about it in http://www.itl.nist.gov/div898/handbook/eda/section3/eda366i.htm.

Example:

What is the probability for finding no fault in a sample of 15 objects drawn from a batch of 300 wherein you expect 3% defective objects overall? This will tell you:

.03 STO I **15** STO J **0** PROB Binom: Binom▲

… returning 0.633 – so the odds are almost two out of three that you will not detect any defect in your sample! [163]

Geom: ***Geometric*** *distribution*:

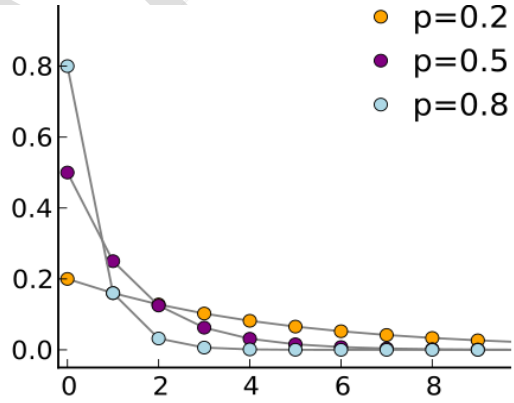
GEOM$_P$ returns

$$p_{Ge}(n)=p_0(1-p_0)^n$$

GEOM▲ returns

$$F_{Ge}(m)=1-(1-p_0)^{m+1},$$

being the probability for a first success after ***m*** = ***x*** *Bernoulli* experiments. The probability ***p₀*** for a success in each such experiment must be specified in $\mathbf{I}$.

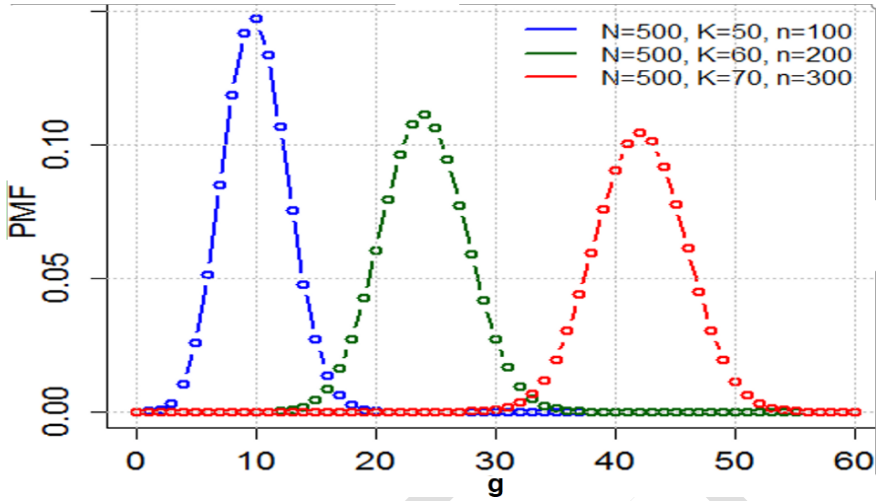


Look up here for more: http://en.wikipedia.org/wiki/Geometric_distribution .

Hyper: ***Hypergeometric*** *distribution* with the *number of successes* ***g*** in $\mathbf{X}$, *gross probability of a success* ***p₀*** in $\mathbf{I}$, *sample size* ***n*** in $\mathbf{J}$, and batch size ***n₀*** in $\mathbf{K}$ (in the diagram shown here are K / N = ***p₀*** and N = ***n₀***).

[163] The exact result for said boundary conditions is 0.626, calculated using the *hypergeometric distribution* below. For decades, such probabilities were computed applying the *binomial* or even the *Poisson distribution* since calculations are easier there. Don't be afraid – you own a *WP43*!

DRAFT



HYPER_p returns $p_H(g; n; p_0; n_0) = \dfrac{\binom{n_0 p_0}{g}\binom{n_0(1-p_0)}{n-g}}{\binom{n_0}{n}}$

(cf. COMB on p. 268 for explanation of the notation).

While the *binomial distribution* assumes that each sample part is returned to the batch after checking, the *hypergeometric distribution* lets you keep your samples out of the batch. This is found more often in real life. It may be neglected, however, in alleged 'large' batches ($n_0 > 10$) and for 'small' sample sizes (<10% of n_0). Check the differences. Start reading here for more: http://en.wikipedia.org/wiki/Hypergeometric_distribution .

NBin: ***Negative binomial*** *distribution* with the *total number of failures* ***f*** (in ***n*** draws) in $\mathbf{X}$, the *gross probability of a success in a single draw* $\boldsymbol{p_0}$ in $\mathbf{I}$, and ***n*** in $\mathbf{J}$. NBIN_P returns

$$p_{NB}(f;\ n;\ p_0) = \binom{n-1}{f-1}\ p_0^f\ (1-p_0)^{n-f} = C_{n-1;f-1}\ p_0^f\ (1-p_0)^{n-f}$$

(cf. COMB on p. 268 and compare BINOM).

Start reading here for more:
http://en.wikipedia.org/wiki/Negative_binomial_distribution .

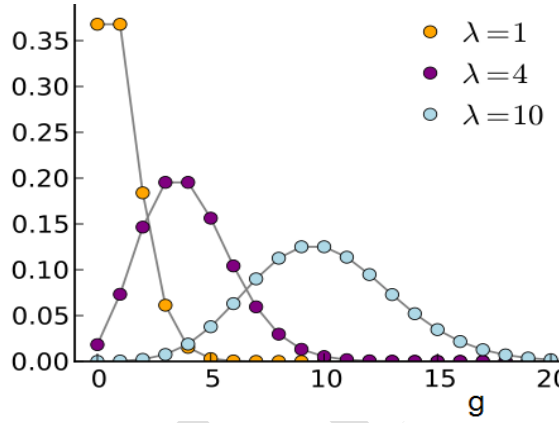
Poiss: ***Poisson*** *distribution* with the *number of successes* ***g*** in $\mathbf{X}$ and the *Poisson parameter* $\boldsymbol{\lambda}$ in $\mathbf{J}$.

DRAFT

POISS$_P$ computes

$$p_P(g;\lambda) = \frac{\lambda^g}{g!}\, e^{-\lambda}$$

and POISS returns the corresponding *CDF* for the maximum number of successes ***m*** in **X**.

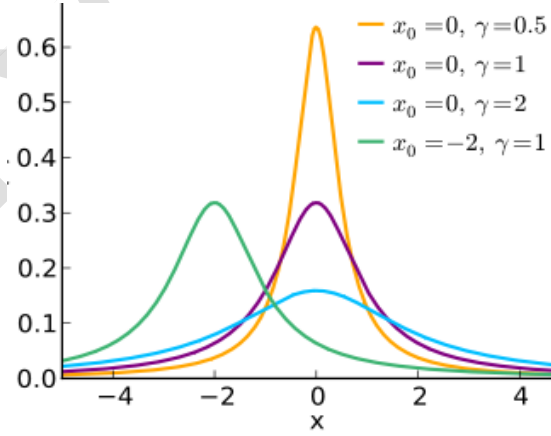


The *Poisson distribution* provides the mathematically simplest model for industrial sampling tests: use $\lambda = np_0$ with the gross error probability p_0 and the sample size ***n*** (cf. BINOM). For the problem introduced with BINOM above, POISS returns 0.638.

Read here for more:
http://www.itl.nist.gov/div898/handbook/eda/section3/eda366j.htm .

Continuous distributions:

Cauch: ***Cauchy-Lorentz*** *distribution* (a.k.a. *Lorentz* or *Breit-Wigner distribution*) with the peak *location* x_0 specified in **I** and the *shape* γ in **J**.



CAUCH$_P$ returns $$f_{Ca}(x) = \left\{\pi\gamma\left[1 + \left(\frac{x - x_0}{\gamma}\right)^2\right]\right\}^{-1}$$

CAUCH$_{\Delta}$ returns $$F_{Ca}(x) = \frac{1}{2} + \frac{1}{\pi}\, arctan\left(\frac{x - x_0}{\gamma}\right)$$

DRAFT

CAUCH^{-1} returns $F_{Ca}^{-1}(p) = x_0 + \gamma \, tan\left[\pi \left(p - \frac{1}{2}\right)\right]$

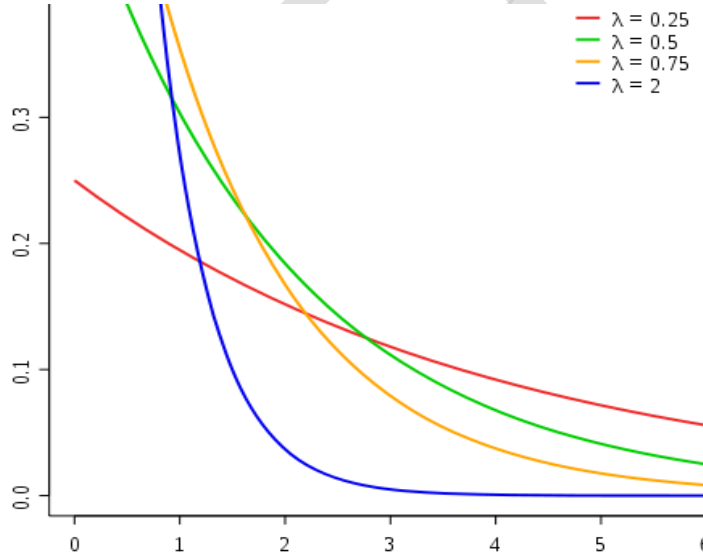
This distribution is quite popular in physics. It is a special case of *Student's t distribution*.

Start reading here for more: *http://en.wikipedia.org/wiki/Cauchy_distribution*.

Expon: ***Exponential*** *distribution* with the *rate* λ in $\mathbf{I}$.

EXPON$_P$ returns $f_{Ex}(x) = \lambda \, e^{-\lambda x}$.

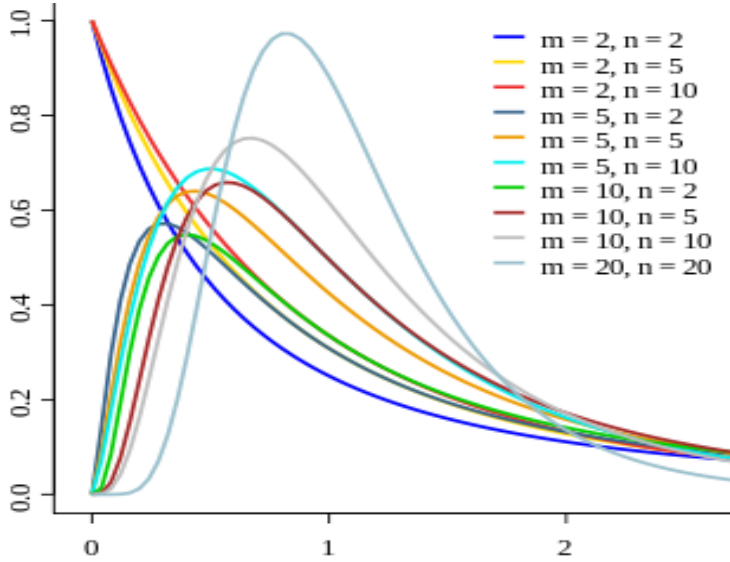
EXPON$_{\Delta}$ returns $F_{Ex}(x) = 1 - e^{-\lambda x}$.



Look up here for more: *http://www.itl.nist.gov/div898/handbook/eda/section3/eda3667.htm*

F(x): ***Fisher's F*** *distribution* with the *degrees of freedom* in $\mathbf{I}$ and $\mathbf{J}$. It is used, e.g., for analyses of variance (*ANOVA*).

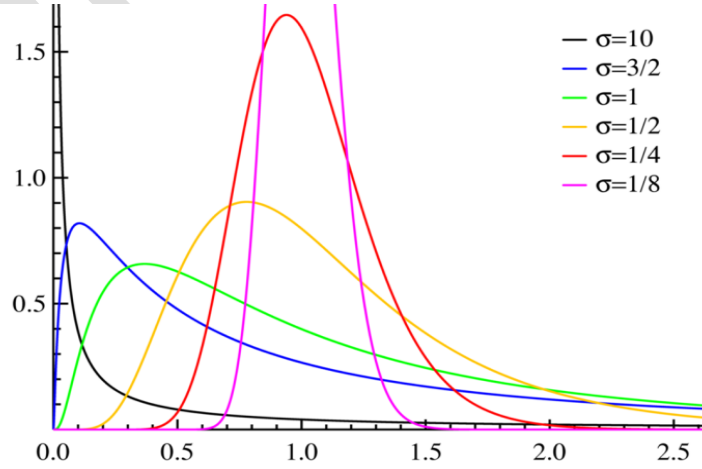
The diagram presents the *PDFs* plotted for different *degrees of freedom* ***m*** *and* ***n*** corresponding to ***i*** and ***j***. – Read here for more: *http://www.itl.nist.gov/div898/handbook/eda/section3/eda3665.htm*



LgNrm: ***Log-normal*** *distribution* with the parameters $\mu = \ln \bar{x}_g$ in **I** and $\sigma = \ln \varepsilon$ in **J** (see some *PDF* plots for $\boldsymbol{\mu} = 0$ below).

LGNRM$_P$ returns $f_{Ln}(x) = \dfrac{1}{x\,\sigma\sqrt{2\pi}}\, e^{-\frac{[ln(x)-\mu]^2}{2\sigma^2}}$

LGNRM$_{\Phi}$ returns $F_{Ln}(x) = \Phi\left(\dfrac{ln(x)-\mu}{\sigma}\right)$ with $\Phi(z)$ denoting the *standardized normal CDF* as presented on p. 275.



DRAFT

Read here for more:
http://www.itl.nist.gov/div898/handbook/eda/section3/eda3669.htm

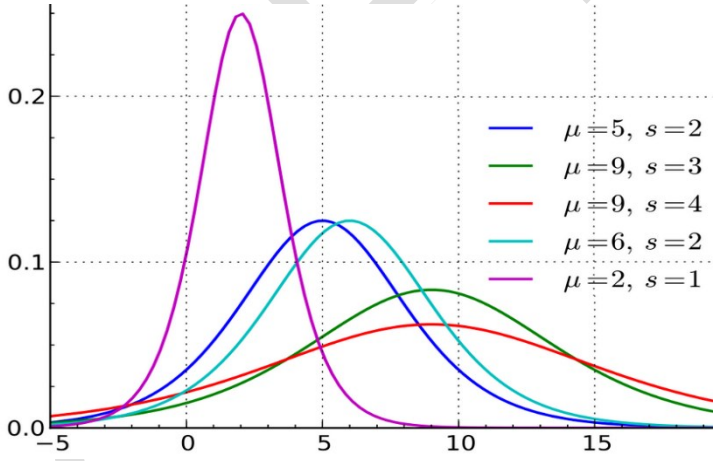
Logis: ***Logistic*** *distribution* with an arbitrary *mean* **μ** given in **I** and a *scale parameter* **s** in **J**.

Substituting $\xi = {}^{x-\mu}/_{s}$,

LOGIS$_P$ returns $f_{Lg}(x) = \dfrac{e^{-\xi}}{(1+e^{-\xi})^2\, s}$

LOGIS$_{\blacktriangle}$ returns $F_{Lg}(x) = {}^{1}/_{(1+e^{-\xi})}$

LOGIS^{-1} returns $F_{Lg}^{-1}(p) = \mu + s\, ln\left(\dfrac{p}{1-p}\right)$

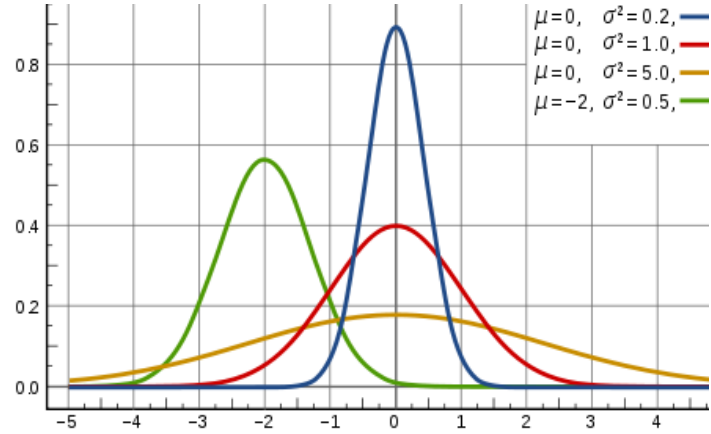


Start reading here for more:
http://en.wikipedia.org/wiki/Logistic_distribution .

Norml: ***Normal*** *distribution* with a *mean* **μ** given in **I** and an *SD* **σ** in **J**.

The red NORML$_P$ returns $f_N(x) = \dfrac{1}{\sigma\sqrt{2\pi}}\, e^{-\frac{(x-\mu)^2}{2\,\sigma^2}} = \varphi\left(\dfrac{x-\mu}{\sigma}\right)$

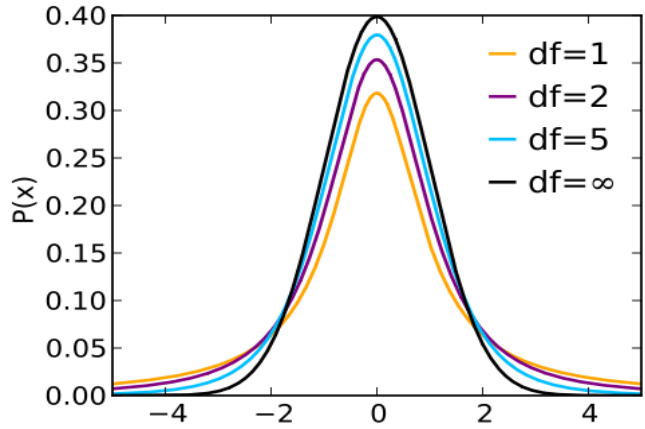
And NORML$_{\blacktriangle}$ returns $F_N(x) = \Phi\left(\dfrac{x-\mu}{\sigma}\right)$ with $\Phi(z)$ denoting the *standardized normal CDF* (cf. the *error function* on p. 316).



Read here for more: http://www.itl.nist.gov/div898/handbook/eda/section3/eda3661.htm

t(x): Standardized ***Student's t*** *distribution* with its *degrees of freedom* (***df***) in **I**.

It is used for hypothesis testing and calculating confidence intervals, e.g., for means. The diagram shows its *PDFs* plotted for different *degrees of freedom*. For ***df*** → ∞ , the shoulders of *t(x)* shrink and it approaches the *standardized normal distribution* (compare the red curve for NORML above). – Read here for more: http://www.itl.nist.gov/div898/handbook/eda/section3/eda3664.htm



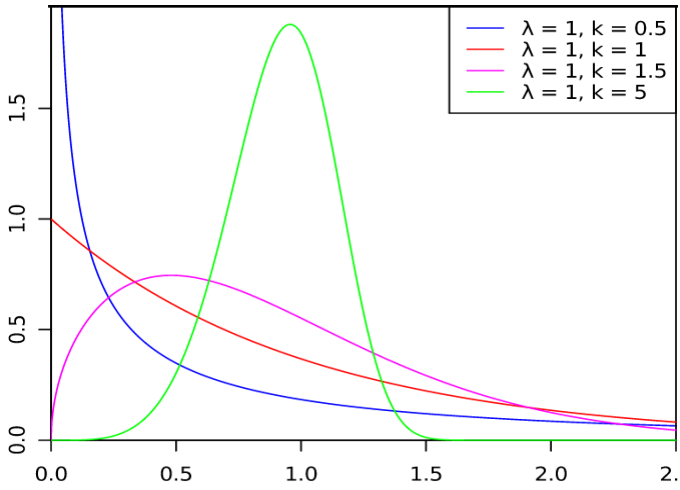
Weibl: ***Weibull*** *distribution* with its *shape parameter* ***b*** in **I** and its *characteristic lifetime* ***T*** in **J**.

WEIBL$_P$ returns $f_W(t) = \frac{b}{T} \cdot \left(\frac{t}{T}\right)^{b-1} e^{-(t/T)^b}$ for t ≥ 0, else 0.

This is a very flexible function – see some curves plotted overleaf (with $k = b$ and $\lambda = 1/T$).

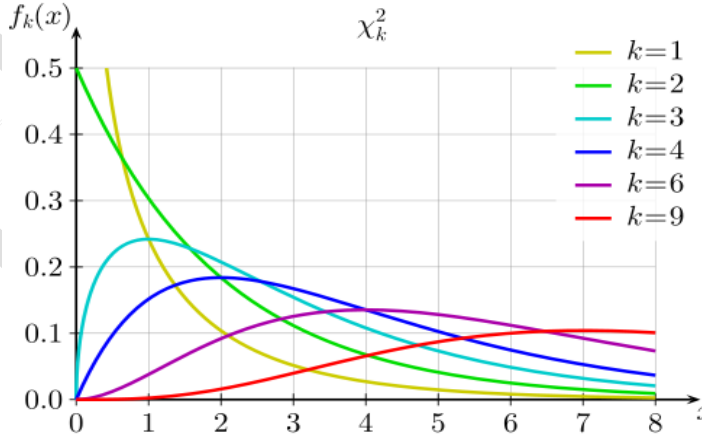
WEIBL returns $F_W(t) = 1 - e^{-(t/T)^b}$

The *Weibull distribution* is widely used, e.g., for analyzing lifetimes of tools or products.



Read here for more: http://www.itl.nist.gov/div898/handbook/eda/section3/eda3668.htm
You may even find some more application fields mentioned in https://en.wikipedia.org/wiki/Weibull_distribution#Applications .

χ^2(x): ***Chi-square*** *distribution* with its *degrees of freedom* given in **I**. It is used for calculating confidence intervals for *SDs, variances, process and machine capabilities*, and the like. The diagram shows *PDF's* for different *degrees of freedom*.



Read here for more: http://www.itl.nist.gov/div898/handbook/eda/section3/eda3666.htm

DRAFT

More Statistical Formulas, also for Curve Fitting

The following equations are for data measured at samples of n specimens (i.e. n is the *sample size*). Note that a complete measurement result must include both: information about the expected value and about its uncertainty.

- For samples drawn out of a *normally* distributed (additive) process, the expected value is the *arithmetic mean* (or *average*) of the sample values and its uncertainty is given by its *standard error* (see $\overline{x}$ and s_m).
- For samples drawn out of a *log-normally* distributed (multiplicative) process, the expected value is the *geometric mean* and its uncertainty is given by its *scattering factor* (see $\overline{x}_g$ and ε_m).
- For samples drawn out of other kinds of processes other measures apply.

Generally, the statistical model shall be chosen that matches observations best – within their statistical errors.[164] Be assured not everything is *normal* (*Gaussian*) in real world.[165] Process characteristics can be detected (and should be checked well in advance of calculating, e.g., means) using suitable tests – turn to applicable statistical reference literature.

The following functions as named in the left column (sorted alphabetically) are all found in STAT or FIT. All sums Σ and products Π in this chapter run from **1** to $\boldsymbol{n}$: [166]

[164] In real life, drastic deviations from the (usually *normal*) model distribution are often found – then you can't expect the calculated consequences matching reality any better, though.

As mentioned in *OMS 2*, we recommend you look deeply into statistics textbooks to ensure you fully understand what you do with the functions provided in your *WP43*. Remember also f. 162.

[165] The *PDF* of a *normal* (a.k.a. *Gaussian*) *distribution* will never reach zero. So this statistical model tells you to expect individual items far, far away from the mean value when your sample becomes large enough. This, however, does not match reality. So we must conclude nothing at all is really *Gaussian* in real world. Nevertheless, the *Gaussian distribution* is a very successful model describing a lot of real-world observations very well. Just never forget the limits of such models.

[166] Diagrams in this chapter were created using either vintage *HP* manuals of the 1970's or (if drawn black on white) the book *Curve Fitting for Programmable Calculators* by *W. M. Kolb*, 1st ed., 1982.

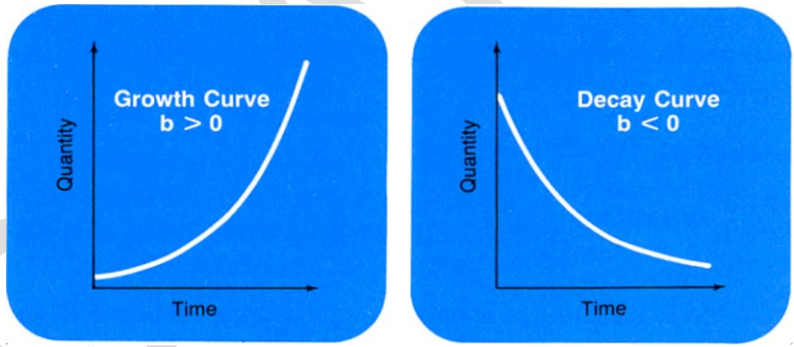
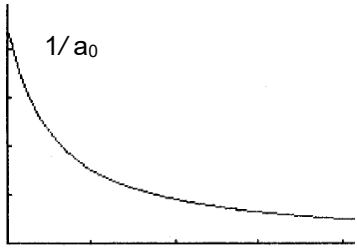
DRAFT

Name	Remarks (cf. pp. 11ff for general information)
CauchF	Selects a *Cauchy* (a.k.a. *Lorentz*, *Breit-Wigner*) peak fit model $$R(x) = \frac{1}{a_0\,(x + a_1)^2 + a_2}$$ for *OLS*.[167] Cf. p. 272 for shapes of such peaks.
CORR (**r**)	For any set of data points (x_i, y_i), the *coefficient of correlation* is $r = {}^{s_{xy}}/_{s_x\, s_y}$. See s_{xy}, s, and dedicated formulas for r or r^2 in the various fit models on pp. 286ff. For an arbitrary fit model $R(x)$, $$r^2 = 1 - \frac{\sum\,[R(x_i) - y_i]^2}{\sum\,(\bar{y} - y_i)^2}$$ is its *coefficient of determination* (indicating the fraction of the variation of the dependent data y determined by the variation of the independent data x). For $r^2 = 1$, y is fully determined by x; for $r^2 = 0$, y is completely independent of x; and, e.g., $r^2 = 0.85$ means 85% of the variation of y is due to x. Note BESTF picks the fit model showing the maximum r^2 out of the models you allowed. A two-parameter regression (the majority of the fit models provided on your *WP43*) is called being (statistically) *significant* at a 99% *confidence level* if $$\sqrt{\frac{r^2}{1 - r^2}\,(n - 2)} \;>\; t_{n-2}^{-1}(0.99)$$ with the right side being the inverse of the *t distribution* for the *degrees of freedom* $n - 2$ (cf. p. 276).

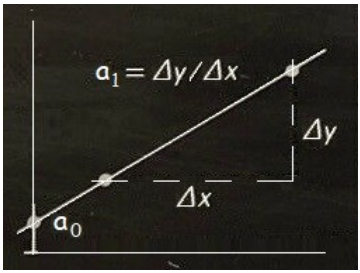
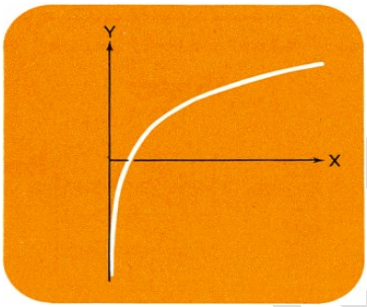
[167] Note that an *ordinary least squares regression* (*OLS*) works best for (horizontal) ***x*** being the independent and (vertical) ***y*** being the dependent variable. In any case, data point errors in direction ***y*** shall be significantly greater than those in direction ***x***. Swap axes if the situation differs in your problem. For an *OLS*, the sum of the squared (vertical) ***y***-distances from the data points to the fit line will be minimized.

See ORTHOF below for a different regression model (*TLS*) and next chapter for more about all the curve fit models provided on your *WP43*.

DRAFT

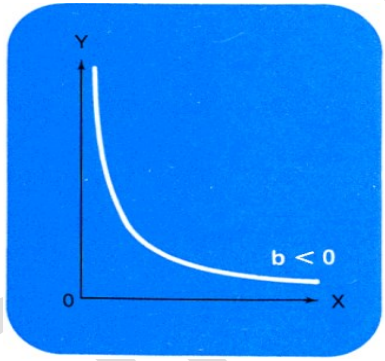
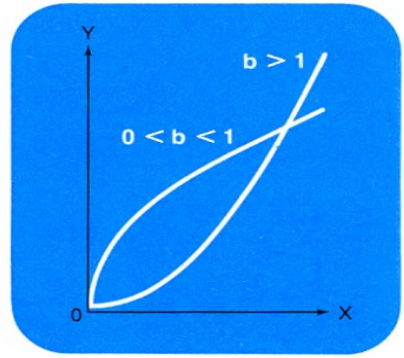
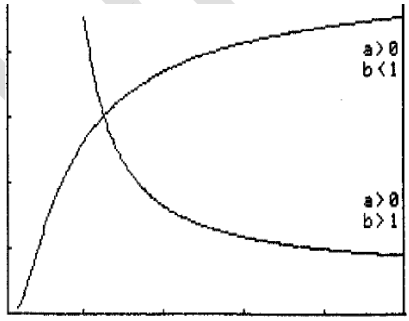
Name	Remarks (cf. pp. 11ff for general information)
COV, **s_{xy}**	For any set of data points (x_i, y_i), the *population covariance* is $$COV_{xy} = \frac{1}{n^2}\left(n\sum x_i y_i - \sum x_i \sum y_i\right)$$ and the *sample covariance* is $$s_{xy} = \frac{1}{n(n-1)}\left(n\sum x_i y_i - \sum x_i \sum y_i\right).$$ Note covariances check linear dependencies only (see the plots *https://de.wikipedia.org/wiki/Stichprobenkovarianz#/media/Datei:CovarianceCorrelation.svg*).
ExpF	Selects the exponential curve fit model $R(x) = a\,e^{bx} = a_0 e^{a_1 x}$ for *OLS*.[167] Generally, this will be a good choice if the measured data follow the shape of one of the two curves pictured (think of, e.g., human population growth or nuclear decay). 
GaussF	Selects a *Gauss* peak fit model $R(x) = a_0\, e^{\frac{(x-a_1)^2}{a_2}}$ for *OLS*.[167] Cf. p. 275 for shapes of such peaks.
HypF	 Selects the hyperbolic fit model $R(x) = {}^{1}/_{(a_0 + a_1 x)}$ for *OLS*.[167]

DRAFT

Name	Remarks (cf. pp. 11ff for general information)
LinF	 Selects the linear fit model $R(x) = a_0 + a_1 x$ for *OLS* (cf. p. 279). Generally, this will be a good choice if the measured data follow a straight line, raising or falling (but compare ORTHOF below).[168]
LogF	 Selects the logarithmic curve fit model $R(x) = a_0 + a_1 ln(x)$ for *OLS* (cf. p. 279). Generally, this will be a good choice if the measured data follow a curve looking like drawn at left.
L.R.	Uses the fit model selected and computes the $\boldsymbol{m}$ (i.e. 2 or 3) parameters of the regression curve through the $\boldsymbol{n}$ data points accumulated. Note $\boldsymbol{m}$ must not exceed $\boldsymbol{n}$. For all fit models provided on your *WP43*, a regression parameter is (statistically) *significant* at a 99% *confidence level* if $$\left.\lvert a_i \rvert\right/_{s(a_i)} > t_{n-m}^{-1}(0.995)\,,$$ with the right side being the inverse of the *t distribution* for $\boldsymbol{n} - \boldsymbol{m}$ *degrees of freedom* (cf. p. 276).
OrthoF	Selects a linear fit model $R(x) = a_0 + a_1 x$ like LINF, but assumes data point errors both in ***x*** and ***y*** being of (about) equal size. The sum of the squared (orthogonal) distances from the data points to the fit line will be minimized. Hence, this model is called *orthogonal regression* (or *total least squares*, *TLS*; a.k.a. *Deming regression*; compare p. 279). The *SD* of the data points from the fit line is

[168] If you want to detect if there is more than just a single linear relation underlying your experimental data (i.e. if the slope changes), look at this: https://xkcd.com/2701/

DRAFT

Name	Remarks (cf. pp. 11ff for general information)
	$$s_o = s_x \; s_y \sqrt{\frac{1-r^2}{s_x^2 + r^2 \, s_y^2}}$$ s_0 equals **s_mi** if ≥ 30 points were generated as described in the application examples in *OMS 2.* See also pp. 286ff and 295ff.
ParabF	Selects a parabolic fit model $R(x) = a_0 + a_1 x + a_2 x^2$ for *OLS* (cf. p. 279).
PowerF	Selects the power curve fit model $R(x) = a \, x^b = a_0 \, x^{a_1}$ for *OLS* (cf. p. 279). This will be a good choice if measured data follow the shape of one of the curves pictured here (look for the *Tower of Pisa* in the *OM*).  
RootF	 Selects the root curve fit model $R(x) = a \, b^{1/x} = a_0 \, a_1^{1/x}$ for *OLS* (cf. p. 279).

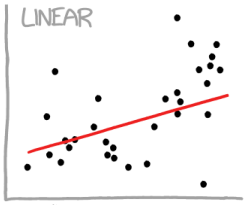
Name	Remarks (cf. pp. 11ff for general information)
s, **s_m**	The ***sample** SD **s*** is the positive square root of the *variance* of a **normally** distributed sample: $$s_x^2 = \frac{n\sum x_i^2 - (\sum x_i)^2}{n(n-1)} = \frac{\sum x_i^2 - n\,\bar{x}^2}{n-1}$$ And the *standard error* (i.e. the *SD* of the *mean* $\bar{x}$) is $s_m = {}^{s}/_{\sqrt{n}}$
s_{mi}	The *SD of the measuring instrument* as measured using the method shown on pp. 295ff. Cf. also ORTHOF and *OMS 2*.
s_w, **s_{mw}**	The ***sample** SD* for weighted or grouped data (where the weight or frequency y_i of each data point x_i was entered via [**Σ+**]) is $$s_w = \sqrt{\frac{\sum y_i \sum y_i x_i^2 - (\sum y_i x_i)^2}{\sum y_i\,(\sum y_i - 1)}}$$ And the *standard error* (i.e. the *SD* of the *weighted mean* $\bar{x}_W$) is $$s_{mw} = {}^{s_w}\Big/_{\sqrt{\sum y_i}}$$
s_{xy}	Cf. COV on p. 280.
$\hat{x}$	Forecast for ***x*** for given ***y*** from the fit model used, see next chapter.
$\bar{x}$	The ***arithmetic** mean* of **normally** distributed data is $\bar{x} = \frac{1}{n}\sum x_i$
$\bar{x}_G$	The ***geometric** mean* of **log-normally** distributed data is $\bar{x}_G = \sqrt[n]{\prod x_i} = exp\left[\frac{1}{n}\sum ln(x_i)\right]$
$\bar{x}_H$	The ***harmonic** mean* is calculated as $\bar{x}_H = {}^{n}\Big/_{\sum \frac{1}{x_i}}$

DRAFT

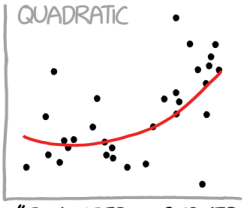
Name	Remarks (cf. pp. 11ff for general information)
$\bar{x}_{RMS}$	The ***quadratic*** *mean* (*root mean squared*) is calculated as $\bar{x}_{RMS} = \sqrt{\frac{1}{n}\sum x_i^2}$
$\bar{x}_w$	The *arithmetic mean* of **normally** distributed <u>weighted</u> or grouped data (where the weight or frequency y_i of each data point x_i was entered via [**Σ+**]) is $\bar{x}_w = \sum x_i y_i \Big/ \sum y_i$
$\hat{y}$	Forecast for ***y*** for given ***x*** from the fit model used, see next chapter.
ε	The *scattering factor* ε_x for a **sample** of **log-normally** distributed data is $ln(\varepsilon_x) = \sqrt{\frac{\sum ln^2(x_i) \; - 2n \; ln \; (\bar{x}_G)}{n-1}}$ Compare s.
ε_m	The *scattering factor* of the *geometric mean* is $\varepsilon_m = \varepsilon^{1/\sqrt{n}}$ Compare s_m.
ε_p	The *scattering factor* ε_p for a **population** of **log-normally** distributed data is $ln(\varepsilon_P) = \sqrt{\frac{n-1}{n}} \; ln(\varepsilon)$ Compare σ.
σ	The **population** *SD* of **normally** distributed data is $\sigma = \sqrt{\frac{n-1}{n}} \; s$
σ_w	The **population** *SD* of <u>weighted</u> or grouped data (where the weight or frequency y_i of each data point x_i was entered via [**Σ+**]) is $\sigma_w = \sqrt{\frac{\sum y_i \; (x_i - \bar{x}_w)^2}{\sum y_i}}$

DRAFT

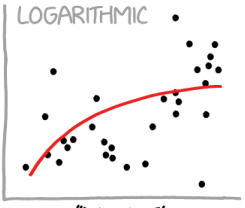
CURVE-FITTING METHODS
AND THE MESSAGES THEY SEND



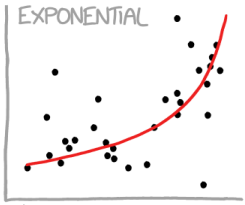
"HEY, I DID A REGRESSION."



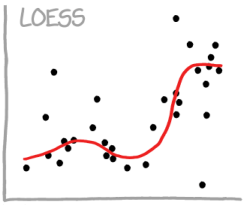
"I WANTED A CURVED LINE, SO I MADE ONE WITH MATH."



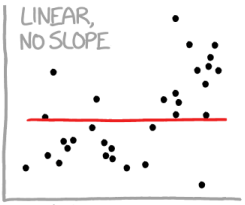
"LOOK, IT'S TAPERING OFF!"



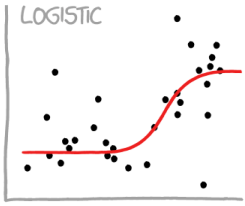
"LOOK, IT'S GROWING UNCONTROLLABLY!"



"I'M SOPHISTICATED, NOT LIKE THOSE BUMBLING POLYNOMIAL PEOPLE."

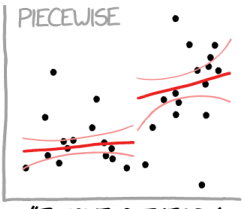


"I'M MAKING A SCATTER PLOT BUT I DON'T WANT TO."

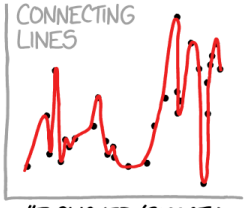


"I NEED TO CONNECT THESE TWO LINES, BUT MY FIRST IDEA DIDN'T HAVE ENOUGH MATH."

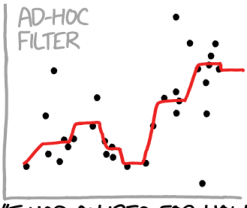
"LISTEN, SCIENCE IS HARD. BUT I'M A SERIOUS PERSON DOING MY BEST."



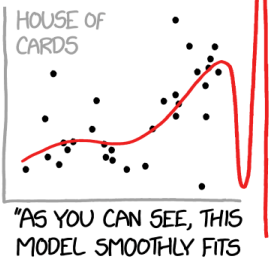
"I HAVE A THEORY, AND THIS IS THE ONLY DATA I COULD FIND."



"I CLICKED 'SMOOTH LINES' IN EXCEL."



"I HAD AN IDEA FOR HOW TO CLEAN UP THE DATA. WHAT DO YOU THINK?"



"AS YOU CAN SEE, THIS MODEL SMOOTHLY FITS THE– *WAIT NO NO DON'T EXTEND IT AAAAAA!!*"

Curve Fitting Models Provided

A proper linear regression is computed for LINF and ORTHOF only. For the other three standard models (EXPF, LOGF, and POWERF) the method of LINF is applied to transformed data. Your data might follow a straight line if you plot ...

- the logarithm of your *y*-data over your *x*-data (then EXPF will fit);
- the logarithm of your *y*-data over the logarithm of your *x*-data (then POWERF will fit);
- your *y*-data over the logarithm of your *x*-data (then LOGF will fit).

Furthermore, your data might follow a straight line if you plot the inverse of your *y*-data over your *x*-data (then HYPF will fit). And there is more below.[169]

This is what your *WP43* will do when you enter statistical data points and compute the parameters of a fit curve:

1. For each new data point, it puts $\boldsymbol{x}$ and $\boldsymbol{y}$ in a new row of STATS, updates the 23 sums listed on pp. 103f (some of them may turn non-numeric for non-positive entries – just do not care), and increments the number $\boldsymbol{n}$. .

2. The subsequent evaluation depends on the fit model you select (cf. pp. 280ff):

   a. **If LINF is chosen** then the *OLS* (cf. p. 279) line parameters a_0 and a_1 are computed following the formulas:

$$a_0 = \frac{\sum x_i^2 \cdot \sum y_i - \sum x_i \cdot \sum x_i y_i}{n\sum x_i^2 - (\sum x_i)^2}, \quad a_1 = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^2 - (\sum x_i)^2} = \frac{s_{xy}}{s_x^2} = r\,\frac{s_y}{s_x}$$

$$\text{with} \quad r = \frac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^2 - (\sum x_i)^2} \cdot \sqrt{n\sum y_i^2 - (\sum y_i)^2}}$$

   The *standard errors* of a_0 and a_1 are calculated using the formulas

$$s(a_1) = \frac{s_y}{s_x}\sqrt{\frac{1-r^2}{n-2}} \quad \text{and} \quad s(a_0) = s(a_1)\sqrt{\frac{n-1}{n}\,s_x^2 + \bar{x}^2} \qquad (*)$$

$$\text{with } s_x^2 = \frac{n\sum x_i^2 - (\sum x_i)^2}{n(n-1)}, \; s_y^2 = \frac{n\sum y_i^2 - (\sum y_i)^2}{n(n-1)}, \; \textit{and } \bar{x} = \frac{1}{n}\sum x_i.$$

[169] The picture on previous page was copied from https://xkcd.com/2048/ .

The forecast for a given $\boldsymbol{x}$ is $\hat{y} = a_0 + a_1\, x$; for a given $\boldsymbol{y}$ it is $\hat{x} = (\, y - a_0)/a_1$.

The *standard error of estimate* (a.k.a. square root of the *residual variances*) is

$$s_{res} = \sqrt{\frac{n-1}{n-2}\left(s_y^2 - a_1^2 s_x^2\right)} = s_y \sqrt{\frac{n-1}{n-2}(1-r^2)} = s(a_1)\, s_x \sqrt{n-1}$$

The *standard error of* a forecast $\hat{y}$ as specified above is

$$s(\hat{y}) = s_{res} \sqrt{\frac{1}{n} + \frac{(x-\bar{x})^2}{\sum (x_i - \bar{x})^2}} = s(a_1) \sqrt{\frac{n-1}{n} s_x^2 + (x-\bar{x})^2}$$

Note this error increases with growing distance from the mean.

b. **If you chose EXPF** then the *OLS* (cf. p. 279) line parameters for the transformed data x_i , $\ln(y_i)$ are computed using

$$a_{0,tEX} = \frac{\sum x_i^2 \cdot \sum \ln(y_i) - \sum x_i \cdot \sum x_i \ln(y_i)}{n \sum x_i^2 - (\sum x_i)^2}$$

$$a_{1,tEX} = \frac{n \sum x_i \ln(y_i) - \sum x_i \sum \ln(y_i)}{n \sum x_i^2 - (\sum x_i)^2}$$

$$r_{tEX} = \frac{n \sum x_i \ln(y_i) - \sum x_i \sum \ln(y_i)}{\sqrt{n \sum x_i^2 - (\sum x_i)^2} \cdot \sqrt{n \sum \ln^2(y_i) - [\sum \ln(y_i)]^2}}$$

The standard errors of $a_{0,tEX}$ and $a_{1,tEX}$ are calculated using the formulas (⋆) on p. 286 but replacing s_y^2 with

$$s_{ty}^2 = \frac{n \sum \ln^2(y_i) - (\sum \ln(y_i))^2}{n(n-1)} \quad \Longrightarrow \quad s(a_{1,tEX}) = \frac{s_{ty}}{s_x} \sqrt{\frac{1 - r_{tEX}^2}{n-2}} \quad \text{here.}$$

The parameters of the fit curve $R(x) = a_0 e^{a_1 x}$ turn out being $a_0 = e^{a_{0,tEX}}$ and $a_1 = a_{1,tEX}$. Thus, $s(a_0) = s\left(a_{0,tEX}\right) e^{a_{0,tEX}} = s\left(a_{0,tEX}\right) a_0$.

The forecast for a given $\boldsymbol{x}$ is $\hat{y} = a_0 e^{a_1 x}$; for a given $\boldsymbol{y}$ it is $\hat{x} = \frac{1}{a_1} \ln\left({}^{y}/_{a_0}\right)$.

DRAFT

c. **If POWERF is chosen** then the *OLS* (cf. p. 279) line parameters for the transformed data $\ln(x_i)$, $\ln(y_i)$ are computed in analogy to the method shown for EXPF. Thus:

$$a_{0,tPO} = \frac{\sum \ln^2(x_i) \cdot \sum \ln(y_i) - \sum \ln(x_i) \cdot \sum \ln(x_i)\ln(y_i)}{n\sum \ln^2(x_i) - [\sum \ln(x_i)]^2}$$

$$a_{1,tPO} = \frac{n\sum \ln(x_i)\ln(y_i) - \sum \ln(x_i)\sum \ln(y_i)}{n\sum \ln^2(x_i) - [\sum \ln(x_i)]^2}$$

$$r_{tPO} = \frac{n\sum \ln(x_i)\ln(y_i) - \sum \ln(x_i)\sum \ln(y_i)}{\sqrt{n\sum \ln^2(x_i) - [\sum \ln(x_i)]^2} \cdot \sqrt{n\sum \ln^2(y_i) - [\sum \ln(y_i)]^2}}$$

The standard errors of $a_{0,tPOW}$ and $a_{1,tPOW}$ are calculated using the formulas (∗) on p. 286 but replacing

$$s_x^2 \text{ with } s_{tx}^2 = \frac{1}{n(n-1)}\left[n\sum \ln^2(x_i) - \left(\sum \ln(x_i)\right)^2\right],$$

$$s_y^2 \text{ with } s_{ty}^2 = \frac{1}{n(n-1)}\left[n\sum \ln^2(y_i) - \left(\sum \ln(y_i)\right)^2\right],$$

$$\text{and } \bar{x} \text{ with } \bar{x}_t = \frac{1}{n}\sum \ln(x_i).$$

The parameters of the fit curve $R(x) = a_0\, x^{a_1}$ turn out being $a_0 = e^{a_{0,tPO}}$ and $a_1 = a_{1,tPO}$. Thus, $s(a_0) = s\left(a_{0,tPO}\right) a_0$.

The forecast for a given $\boldsymbol{x}$ is $\hat{y} = a_0\, x^{a_1}$; for a given $\boldsymbol{y}$ it is $\hat{x} = \sqrt[a_1]{y/a_0}$.

d. **If you chose LOGF** then the *OLS* (cf. p. 279) line parameters for the transformed data $\ln(x_i)$, y_i are computed in analogy to the method shown for EXPF. Thus they are

$$a_{0,tLO} = \frac{\sum \ln^2(x_i) \cdot \sum y_i - \sum \ln(x_i) \cdot \sum y_i \ln(x_i)}{n\sum \ln^2(x_i) - [\sum \ln(x_i)]^2}$$

$$a_{1,tLO} = \frac{n\sum y_i \ln(x_i) - \sum \ln(x_i)\sum y_i}{n\sum \ln^2(x_i) - [\sum \ln(x_i)]^2}$$

$$r_{tLO} = \frac{n\sum y_i \ln(x_i) - \sum \ln(x_i)\sum y_i}{\sqrt{n\sum \ln^2(x_i) - [\sum \ln(x_i)]^2} \cdot \sqrt{n\sum y_i^2 - [\sum y_i]^2}}$$

DRAFT

The standard errors of $a_{0,tLOG}$ and $a_{1,tLOG}$ are calculated using the formulas (*) on p. 286 but replacing

$$s_x^2 \quad \text{with} \quad s_{tx}^2 = \frac{1}{n(n-1)}\left[n\sum \ln^2(x_i) - \left(\sum \ln(x_i)\right)^2\right]$$

$$\text{and} \quad \bar{x} \quad \text{with} \quad \bar{x}_t = \frac{1}{n}\sum \ln(x_i)\,.$$

The parameters of the fit curve $R(x) = a_0 + a_1 ln(x)$ are just $a_0 = a_{0,tLO}$ and $a_1 = a_{1,tLO}$.

The forecast for a given $\boldsymbol{x}$ is $\hat{y} = a_0 + a_1\, ln(x)$; for a given $\boldsymbol{y}$ it is $\hat{x} = \exp\left({}^{y\,-\,a_0}/_{a_1}\right)$.

e. **If HYPF is chosen** then the parameters of the *OLS* (cf. p. 279) curve $\mathrm{R(x)} = {}^{1}/_{(a_0 + a_1 x)}$ are computed to be

$$a_{0,HY} = \frac{\sum x_i^2 \cdot \sum \frac{1}{y_i} - \sum x_i \cdot \sum \frac{x_i}{y_i}}{n\sum x_i^2 - (\sum x_i)^2} \quad \text{and} \quad a_{1,HY} = \frac{n\sum \frac{x_i}{y_i} - \sum x_i \cdot \sum \frac{1}{y_i}}{n\sum x_i^2 - (\sum x_i)^2}$$

$$r_{HY}^2 = \frac{a_{0,HY}\sum \frac{1}{y_i} + a_{1,HY}\sum \frac{x_i}{y_i} - \frac{1}{n}\left(\sum \frac{1}{y_i}\right)^2}{\sum \frac{1}{y_i^2} - \frac{1}{n}\left(\sum \frac{1}{y_i}\right)^2}$$

The forecast for a given $\boldsymbol{x}$ is $\hat{y} = {}^{1}/_{\left(a_{0,HY} + a_{1,HY}\, x\right)}$; for a given $\boldsymbol{y}$ it is $\hat{x} = \left[\left({}^{1}/_{y}\right) - a_{0,HY}\right]/a_{1,HY}$.

f. **If you chose ROOTF** then the parameters of the *OLS* (cf. p. 279) curve are computed using

$$A = n\sum \frac{1}{x_i^2} - \left(\sum \frac{1}{x_i}\right)^2$$

$$B = \frac{1}{A}\left[\sum \frac{1}{x_i^2} \cdot \sum ln(y_i) - \sum \frac{1}{x_i} \cdot \sum \frac{ln(y_i)}{x_i}\right]$$

$$C = \frac{1}{A}\left[n\sum \frac{ln(y_i)}{x_i} - \sum \frac{1}{x_i} \cdot \sum ln(y_i)\right]$$

DRAFT

The parameters of the fit curve $R(x) = a_0\, a_1^{1/x}$ turn out being just $a_{0,\sqrt{\ }} = e^B$ and $a_{1,\sqrt{\ }} = e^C$.

$$r^2_{\sqrt{\ }} = \frac{B\sum ln(y_i) + C\sum \frac{ln(y_i)}{x_i} - \frac{1}{n}[\sum ln(y_i)]^2}{\sum [ln(y_i)]^2 - \frac{1}{n}[\sum ln(y_i)]^2}$$

The forecast for a given $\boldsymbol{x}$ is $\hat{y} = a_{0,\sqrt{\ }} \times a_{1,\sqrt{\ }}^{1/x}$; for a given $\boldsymbol{y}$ it is $\hat{x} = \lg\left(a_{1,\sqrt{\ }}\right)\left[\lg(y) - \lg\left(a_{0,\sqrt{\ }}\right)\right]$.

g. **If PARABF is chosen** then the parameters of the *OLS* (cf. p. 279) curve are computed using

$$A = n\sum x_i^2 - \left(\sum x_i\right)^2, \qquad B = n\sum x_i^2 y_i - \sum x_i^2 \cdot \sum y_i,$$

$$C = n\sum x_i^3 - \sum x_i^2 \cdot \sum x_i, \qquad D = n\sum x_i y_i - \sum x_i \cdot \sum y_i,$$

$$E = n\sum x_i^4 - \left(\sum x_i^2\right)^2$$

Then the parameters of the fit curve $R(x) = a_0 + a_1 x + a_2 x^2$ are

$$a_2 = \frac{A\,B - C\,D}{A\,E - C^2}, \qquad a_1 = \frac{D - a_2 C}{A},$$

$$\text{and} \quad a_0 = \frac{1}{n}\left(\sum y_i - a_2 \sum x_i^2 - a_1 \sum x_i\right).$$

$$\text{And} \quad r^2_{PA} = \frac{a_0 \sum y_i + a_1 \sum x_i y_i + a_2 \sum x_i^2 y_i - \frac{1}{n}(\sum y_i)^2}{\sum y_i^2 - \frac{1}{n}(\sum y_i)^2}$$

The forecast for a given $\boldsymbol{x}$ is $\hat{y} = a_0 + a_1 x + a_2 x^2$; for a given $\boldsymbol{y}$ it is

$$\hat{x}_{1,2} = \frac{1}{2a_2}\left(-a_1 \pm \sqrt{a_1^2 - 4a_2(a_0 - y)}\right).$$

h. **If you chose GAUSSF** then the parameters of the *OLS* (cf. p. 279) curve are computed using the auxiliary terms *A, C,* and *E* exactly as in PARABF. The other terms are

DRAFT

$$B = n\sum x_i^2 \ln(y_i) - \sum x_i^2 \cdot \sum \ln(y_i),$$

$$D = n\sum x_i \ln(y_i) - \sum x_i \cdot \sum \ln(y_i), \quad F = \frac{A\,B - C\,D}{A\,E - C^2}, \quad G = \frac{D - F\,C}{A},$$

$$\text{and} \quad H = \frac{1}{n}\left(\sum ln(y_i) - F\sum x_i^2 - G\sum x_i\right).$$

The parameters of the fit curve $R(x) = a_0\, e^{(x-a_1)^2/a_2}$ are then

$$a_2 = \frac{1}{F}, \quad a_1 = -\frac{G}{2}\,a_2 \quad \text{and} \quad a_0 = e^{\,H - F\,a_1^2}.$$

$$r_{GA}^2 = \frac{H\sum ln(y_i) + G\sum x_i\, ln(y_i) + F\sum x_i^2\, ln(y_i) - \frac{1}{n}[\sum ln(y_i)]^2}{\sum[ln(y_i)]^2 - \frac{1}{n}[\sum ln(y_i)]^2}$$

The forecast for a given $\boldsymbol{x}$ is $\hat{y} = a_0\, e^{(x-a_1)^2/a_2}$; for a given $\boldsymbol{y}$ it is

$$\hat{x}_{1,2} = a_1 \pm \sqrt{a_2 \ln(\,{}^{y}/_{a_0}\,)}$$

i. **If CAUCHF is chosen** then the parameters of the *OLS* (cf. p. 279) curve are computed using the auxiliary terms *A, C,* and *E* exactly as in PARABF. The other terms are

$$B = n\sum \frac{x_i^2}{y_i} - \sum x_i^2 \cdot \sum \frac{1}{y_i} \quad \text{and} \quad D = n\sum \frac{x_i}{y_i} - \sum x_i \cdot \sum \frac{1}{y_i}.$$

F and *G* are calculated as for GAUSSF but with the components computed here; and

$$H = \frac{1}{n}\left(\sum \frac{1}{y_i} - G\sum x_i - F\sum x_i^2\right).$$

The fit curve $R(x) = {}^{1}/_{[a_0(x+a_1)^2 + a_2]}$ is specified by

$$a_0 = F, \quad a_1 = \frac{G}{2a_0}, \quad \text{and} \quad a_2 = H - F\,a_1^2.$$

$$r_{CA}^2 = \frac{H\sum \frac{1}{y_i} + G\sum \frac{x_i}{y_i} + F\sum \frac{x_i^2}{y_i} - \frac{1}{n}\left(\sum \frac{1}{y_i}\right)^2}{\sum \left(\frac{1}{y_i}\right)^2 - \frac{1}{n}\left(\sum \frac{1}{y_i}\right)^2}$$

DRAFT

The forecast for a given x is $\hat{y} = {}^{1}/_{[a_0(x+a_1)^2+a_2]}$; and the forecast for a given y is

$$\hat{x}_{1,2} = -a_1 \pm \sqrt{\frac{1}{a_0}\left(\frac{1}{y} - a_2\right)}$$

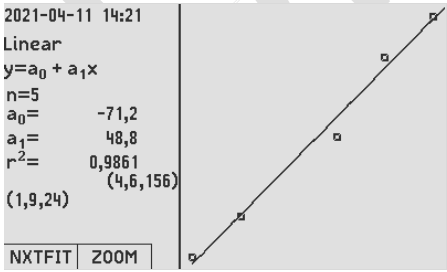
j. **If you chose BESTF** then the correlation coefficient will be computed with your data for model **a** and with the transformed data for models **b** through **i**, if allowed (cf. the *IOI*). The model delivering the greatest r^2 will be selected as best fitting model.

k. **If you chose ORTHOF** then the total least squares regression line parameters a_0 and a_1 will be computed following the formulas

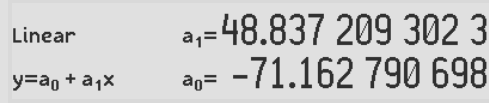
$$a_1 = \frac{1}{2s_{xy}}\left[s_y^2 - s_x^2 + \sqrt{\left(s_y^2 - s_x^2\right)^2 + 4s_{xy}^2}\right] \quad \text{and} \quad a_0 = \bar{y} - a_1\bar{x}$$

assuming the data point errors in direction y and x are about equal. The other formulas for ORTHOF can be taken over from LINF.

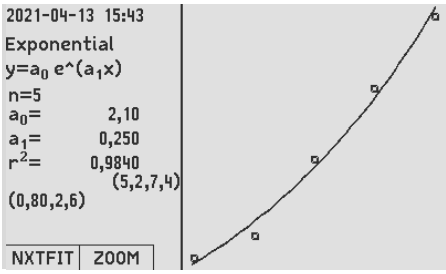
The outputs of ASSESS (left, to scale) **and L.R.** are generally formatted as shown below, allowing for up to 11 digits displayed for each model parameter (far more than usually significant). The following outputs are just dummies for demonstration – some fit models may not make much sense for the sample data used:



Note the values in parentheses bottom left in the ASSESS display are the maximum and minimum of the plot area in format (***x***,***y***).

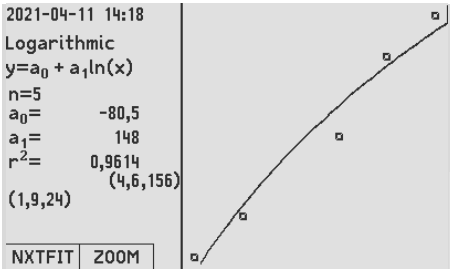


DRAFT



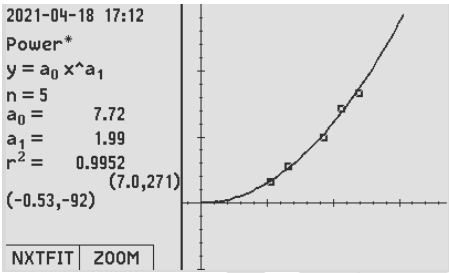
Exponential $a_1 = 2.500\,724\,76 \times 10^{-1}$

$y = a_0\, e^{(a_1 x)}$ $a_0 = 2.102\,978\,249\,05$



Logarithmic $a_1 = 148.222\,520\,145$

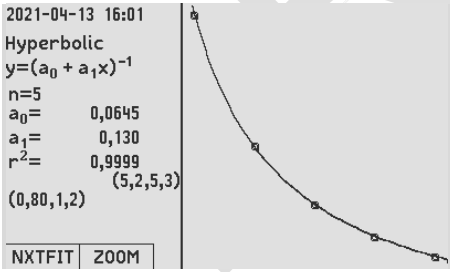
$y = a_0 + a_1 \ln(x)$ $a_0 = -80.532\,162\,769$



If zero is included in the drawing range (which may also be achieved by zooming out), the respective axis or axes will be drawn (see also Cauchy and Gauss).

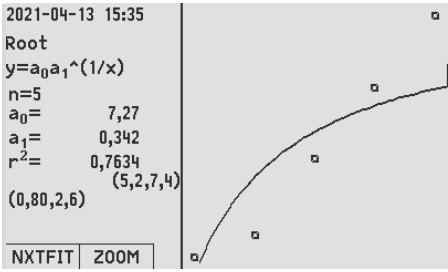
Power $a_1 = -8.052\,852\,4 \times 10^{-1}$

$y = a_0\, x^{a_1}$ $a_0 = 5.229\,414\,822\,53$



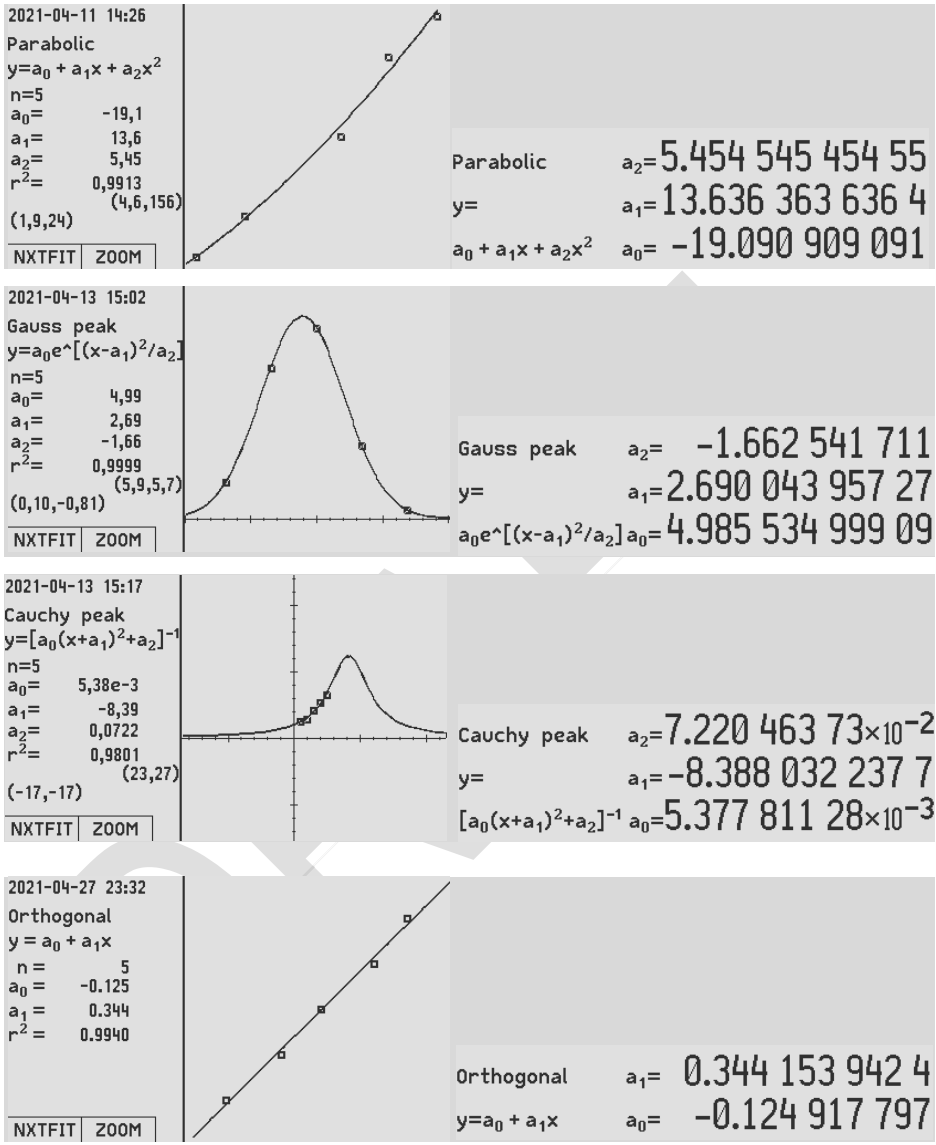
Hyperbolic $a_1 = 1.302\,069\,21 \times 10^{-1}$

$y = (a_0 + a_1 x)^{-1}$ $a_0 = 6.452\,434\,35 \times 10^{-2}$



Root $a_1 = 0.341\,758\,18$

$y = a_0 a_1^{(1/x)}$ $a_0 = 7.270\,848\,526\,85$

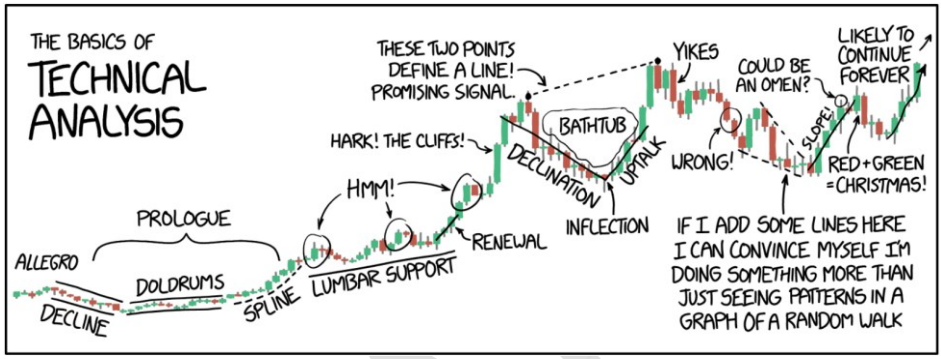


About general polynomial fits to *n* measured data points:

You have seen LINF and PARABF above, being polynomials of grade 1 and 2. For a polynomial $R(x) = a_0 + a_1x + a_2x^2 + \ ... \ + a_gx^g$ of grade ***g*** < ***n***, the system of linear equations determining its parameters looks like

$$Y = X\,A \quad \text{with} \quad Y = \begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix}, \quad X = \begin{pmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^g \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^g \end{pmatrix} \quad \text{and} \quad A = \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_g \end{pmatrix}.$$

Its solution is $A = (X^T X)^{-1} X^T Y$. Your *WP43* will help you solving this matrix equation. Though, do not exaggerate (see https://xkcd.com/2101/):[170]



The Pedestrian's Measurement System Analysis (*MSA*)

Knowing the measuring instrument or system you use is essential for assessing its indications and results (cf. f. 64 on pp. 113f). To measure its actual precision under real-world conditions easily, proceed as explained below:

1. Create a square diagram with equal axes.

   Some fundamentals: Assume a constant object ***A*** and an ideal measuring instrument in constant environment. Measure a particular dimension ***x*** of ***A*** twice independently under unchanged conditions using this instrument; you will get two readings x_1 and x_2. Both will be equal (due to the ideal instrument and constant conditions), so the data point (x_1, x_2) will be located perfectly on the (45°) diagonal of said diagram.

   But all technical processes scatter; real measuring processes do as well (actually, uncertainty is inherent in any physical measurement). Thus,

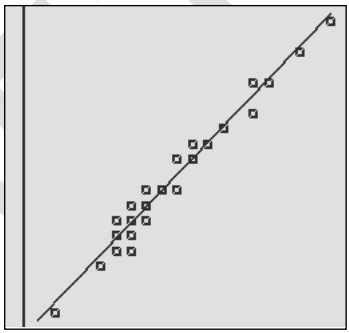
[170] *Science* grows in competition between experiment and theory. Theory interprets observations but also creates predictions in a range where no observations exist so far. Experimental observations may support a theory or falsify it. Did you ever discover a *Technical Analysis* comparing its predictions to what really happened? Think about it.

DRAFT

taking a large number of independent high resolution measurements ***x*** of ***A***, using the same measuring system, combining every two subsequent measurements to one data point (x_1, x_2), and plotting these points will result in a circular cloud of normally distributed points centered on the diagonal. The cloud diameter will be the smaller the better the measuring system is.

Investigating two different, constant objects this way will result in two circular clouds of said kind, three in three, etc. Instead of measuring just very few different objects many times, however, it is advantageous measuring many different objects just two times (the returns will be more diversified and deliver more information). Thus:

2. Perform two successive, independent series of measurements of a set of constant objects A_1 … A_n under at most unchanged conditions using the same instrument.



3. Plot the data points (x_{1j}, x_{2j}) for all A_j in said diagram. They should build an ant trail (see here, cf. the example in *OMS 2*).

4. If its center coincides with the 45° line and the trail is 'narrow enough' then the instrument employed is considered 'sufficiently precise'. For usual narrow trails, (≥) 30 samples are required to achieve a reliable result for the uncertainty of the measuring system by determining the trail width

$$W^2 = s_x^2\, s_y^2\, \frac{1-r^2}{s_x^2 + r^2\, s_y^2} = s_y^2\, \frac{1-r^2}{1+a_1^2} = s_{mi}^2$$

If $30\, s_{mi} \leq \boldsymbol{T}$ then this system is precise enough (a.k.a. *capable*) for controlling a full tolerance band width ***T*** under said conditions.[171]

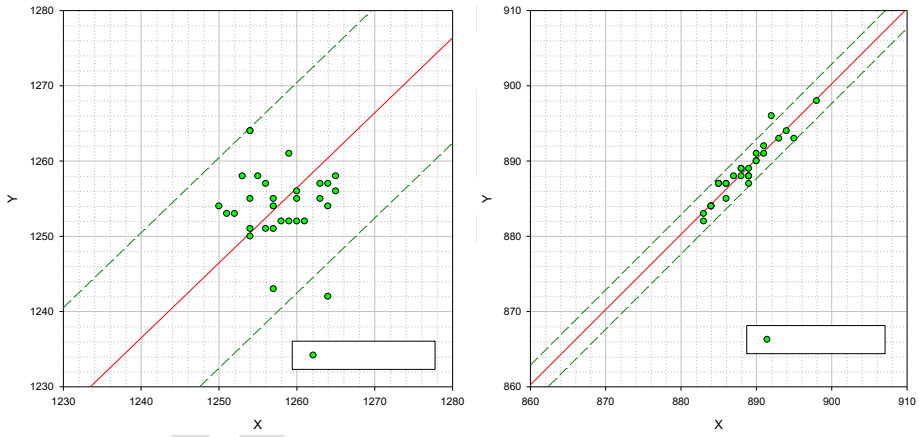
Further analyses of the measuring system investigated are viable by interpreting such diagrams in more detail. They require just plain logic and common sense. Think! Undeniable advantage: parts will never lie to you.

[171] The number 30 appears twice on this page but independently. – Let's emphasize: s_{mi} measured this way reflects the conditions prevalent during the measurements. This is wanted since you strive for knowing a realistic (real-world) precision, e.g., at the place the instrument will be deployed later on. So you do not rely on the values in its data sheet (usually determined under unknown 'ideal' conditions).

If you collect ≥ 30 test objects from a steady state production process, you can determine not only the real-life precision of the measuring system used but also the *SD* of this production process. Then the ratio $(L - W) / W$ (with ***L*** being the trail length) shall be greater than a fixed limit **B** to warrant this measuring system being *capable* to control this production process (cf. *OMS 2*). In automotive industry, **B** = **9.5**. Other industries are less demanding – a minimum **B** = **7.5** shall be achieved.

With many samples approximately uniformly distributed over a certain range the measured points of the overall trail will fill a sausage (i.e. an oval, not an ellipse!).

Plots of this kind may also support you monitoring industrial processes, directly comparing various instruments, settings, etc. Find below, for example, a pair of plots of parts produced on an automated high-speed precision machine set up in two different ways (one small square represents 2 × 2 µm, the dashed lines are $\pm 2\, s_{mi}$ from the center line fitted through the 30 points measured).[172] Evidently the production precision could be improved by a factor 5, applying a relatively simple change in the machine setup:



This method of displaying measured data is very efficient. Although taught in seminars and workshops (and advertised under various names), applicable literature is scarce.[173] Note you can use this analysis method for qualifying any

[172] These diagrams were created using *SigmaPlot*, a software for scientific graphing. I can highly recommend it – it really does what you tell it to do and has some powerful analysis features implemented (see also next footnote).

[173] See, e.g., *https://blog.minitab.com/en/real-world-quality-improvement/orthogonal-regression-testing-the-equivalence-of-instruments*. Actually, our *MSA* diagram grew

measuring system or process (mechanical, electronical, whatever) at any location in the universe – the one and only hard requisite is it shall leave your test samples constant (but it will even tell you when this condition was not fulfilled).

Error Propagation in Calculations

Experimental data always go with errors, caused by, e.g., the uncertainties of the measuring method, the instrument used, and environmental variations (cf. f. 64 on p. 113). Even under carefully controlled environmental and measuring conditions, random errors remain (cf. previous chapter). These errors must be taken into account for a proper estimation of the uncertainties of your results computed using those experimental data. For about 200 years, *Gauss' least squares method* can be employed for this task.

Assume you know that your result *R* depends on ***n*** independent experimental parameters $\boldsymbol{x_1} \ldots \boldsymbol{x_n}$. Each such parameter $\boldsymbol{x_i}$ shows an uncertainty or 'error' $\boldsymbol{\Delta x_i}$. Now, if $R = f(x_1, \ldots, x_n)$ then its total 'error' is

$$\Delta R = f(x_1 \pm \Delta x_1, \ldots, x_n \pm \Delta x_n) - f(x_1, \ldots, x_n)$$

$$= \pm\sqrt{\left(\frac{df}{dx_1}\right)^2 {\Delta x_1}^2 + \cdots + \left(\frac{df}{dx_n}\right)^2 {\Delta x_n}^2}$$

Often, the differential terms under the square root are either unknown or tedious to determine analytically. Luckily, however, this root can be simplified: $\Delta R = \pm\sqrt{{\Delta f_1}^2 + \cdots + {\Delta f_n}^2}$. And with your *WP43*, the following will do for computing $\boldsymbol{\Delta R}$, even if f should be '*very curvy*':

1. Program the function $R = f(x_1, x_2, \ldots, x_n)$ in a way you can vary its parameters easily.

---

out of a *Shainin* workshop about industrial process improvement I attended in 1996.

In four decades of professional experience, I found such correlation diagrams being the most powerful although yet simple tools for easily assessing the quality of real-life processes in arbitrary environments. They ease convincing other people as well, even non-tech. Such plots support decisions far better than staring at numbers. Mankind has got simply far longer experience with pictures (diagrams) than with numbers.

Be very careful when employing *MS Excel* for plotting in this matter: it is unable to draw square diagrams reliably; plots will vary depending on the particular output device used.

DRAFT

2. Let your *WP43* compute $R = f(x_1, x_2, \ldots, x_n)$.
3. Let it compute $R_{1+} = f(x_1 + \Delta x_1, x_2, \ldots, x_n)$ and $\Delta R_{1+} = R_{1+} - R$.
4. Let it compute $R_{1-} = f(x_1 - \Delta x_1, x_2, \ldots, x_n)$ and $\Delta R_{1-} = R_{1-} - R$.
5. Repeat the last two steps for each parameter from x_2 up to x_n.

Being through with all ***n*** parameters, you will eventually get

$$\Delta R = \pm\sqrt{\frac{1}{2}\left(\Delta R_{1+}^2 + \Delta R_{1-}^2 + \Delta R_{2+}^2 + \Delta R_{2-}^2 + \cdots + \Delta R_{n+}^2 + \Delta R_{n-}^2\right)}$$

The summands under the square root have become simple differences which were determined most easily using your *WP43*.

For 'small errors' or less curvature of ***f***, a simpler algorithm will do, requiring down to half as many steps:

1. Program the function $R = f(x_1, x_2, \ldots, x_n)$ in a way you can vary its parameters easily.
2. Let your *WP43* compute $R = f(x_1, x_2, \ldots, x_n)$.
3. Let it compute $R_1 = f(x_1 + \Delta x_1, x_2, \ldots, x_n)$ and $\Delta R_1 = R_1 - R$.
4. Let it compute $R_2 = f(x_1, x_2 + \Delta x_2, \ldots, x_n)$ and $\Delta R_2 = R_2 - R$.
5. Repeat the last step for each parameter from x_3 up to x_n.

Being through with all ***n*** parameters, you will end with

$$\Delta R = \pm\sqrt{\Delta R_1^2 + \Delta R_2^2 + \cdots + \Delta R_n^2}$$

You might know this formula from your university or lab classes.

At the bottom line, we have replaced a formula requiring a lot of symbolic mathematics and few numeric calculations by an equivalent formula requiring no symbolic mathematics at all but many numeric calculations. Programmed, your *WP43* can do all these repetitive calculations for you automatically. Determining derivatives is not required anymore.

DRAFT

Solving Differential Equations (1)

The method applied to the examples in *OMS 3* develops as explained below: [174]

First, we are going to solve a ***1D*** problem of the kind

$$d^2f/dt^2 = a - b\left(df/dt\right)^2$$

This is the equation of motion for a body falling through a medium featuring drag proportional to said body's velocity squared. For earthly problems, take $a = 9.81\frac{m}{s^2} = g$ and $b = \delta/M$ with M being the mass of said body and the constant parameter δ taking care of the viscosity of the medium (e.g., air) as well as size and shape of the falling body as a whole.

For a first guess, let us assume $b = 0$. So there will be no drag at all, the body will be just accelerated by $a = g$. Then for two arbitrary subsequent points in time, the development of

- vertical velocity will be like $\left(\frac{df}{dt}\right)_{i+1} \approx \left(\frac{df}{dt}\right)_i + a\Delta t$ and
- position over ground will be like $f_{i+1} \approx f_i + \left(\frac{df}{dt}\right)_i \Delta t$.

Proceeding from time zero in small, constant time steps $\Delta t = t_{i+1} - t_i$:

$$f_1 \approx f_0 + \left(\frac{df}{dt}\right)_0 \Delta t \quad \text{and} \quad \left(\frac{df}{dt}\right)_1 \approx \left(\frac{df}{dt}\right)_0 + a\Delta t\,,$$

$$f_2 \approx f_1 + \left(\frac{df}{dt}\right)_1 \Delta t \quad \text{and} \quad \left(\frac{df}{dt}\right)_2 \approx \left(\frac{df}{dt}\right)_1 + a\Delta t \text{ , etc.}$$

Using the following, so-called *half-step method*, however, a better approximation of the slope of f is achieved principally:

$$\left(\frac{df}{dt}\right)_{1/2} \approx \left(\frac{df}{dt}\right)_0 + a\frac{\Delta t}{2}\,, \quad \left(\frac{df}{dt}\right)_{i+\frac{1}{2}} \approx \left(\frac{df}{dt}\right)_{i-\frac{1}{2}} + a\Delta t\ ,$$

$$\text{and} \quad f_{i+1} \approx f_i + \left(\frac{df}{dt}\right)_{i+1/2} \Delta t$$

Proceeding from time 0 in small steps Δt again, we get $\left(\frac{df}{dt}\right)_{1/2}$ as above, and

$$f_1 \approx f_0 + \left(\frac{df}{dt}\right)_{1/2} \Delta t\ , \quad \left(\frac{df}{dt}\right)_{3/2} \approx \left(\frac{df}{dt}\right)_{1/2} + a\Delta t\ ,$$

[174] The procedure coarsely follows *R. M. Eisberg, Applied Mathematical Physics with Programmable Pocket Calculators, McGraw-Hill*, New York, 1976.

DRAFT

$$\text{and} \quad f_2 \approx f_1 + \left(\frac{df}{dt}\right)_{3/2} \Delta t\,, \quad \text{etc.}$$

Let us drop the restriction for b now. Replacing a in the previous set of equations by the right side of the differential equation on p. 300, we will get the following new set:

$$\left(\frac{df}{dt}\right)_{1/2} \approx \frac{df}{dt_0} + \left[a - b\left(\frac{df}{dt}\right)_0^2\right] \frac{\Delta t}{2}$$

$$\left(\frac{df}{dt}\right)_{i+1/2} \approx \left(\frac{df}{dt}\right)_{i-\frac{1}{2}} + \left[a - b\left(\frac{df}{dt}\right)_{i-\frac{1}{2}}^2\right] \Delta t$$

$$f_{i+1} \approx f_i + \left(\frac{df}{dt}\right)_{i+1/2} \Delta t$$

Proceeding from time zero in small steps Δt again, we get $\left(\frac{df}{dt}\right)_{1/2}$ as above, as well as

$$f_1 \approx f_0 + \left(\frac{df}{dt}\right)_{1/2} \quad \text{and} \quad \left(\frac{df}{dt}\right)_{3/2} \approx \left(\frac{df}{dt}\right)_{1/2} + \left[a - b\left(\frac{df}{dt}\right)_{1/2}^2\right] \Delta t$$

$$f_2 \approx f_1 + \left(\frac{df}{dt}\right)_{3/2} \Delta t, \quad \text{etc.}$$

This *half-step method* as explained above can be applied easily to all ordinary differential equations of 2nd order which can be written like

$$\frac{d^2 f}{dt^2} = h\left(t;\ f;\ \frac{df}{dt}\right)$$

with an arbitrary *real* function h depending on time, the function itself and its first derivative. The equations applicable in this general case are then

$$\left(\frac{df}{dt}\right)_{1/2} \approx \left(\frac{df}{dt}\right)_0 + h\left(t_0;\ f_0\,;\ \left[\frac{df}{dt}\right]_0\right) \frac{\Delta t}{2}$$

$$\left(\frac{df}{dt}\right)_{i+1/2} \approx \left(\frac{df}{dt}\right)_{i-1/2} + h\left(t_{i-1/2}\,;\ f_{i-1/2}\,;\ \left[\frac{df}{dt}\right]_{i-1/2}\right) \Delta t$$

$$f_{i+1} \approx f_i + \left(\frac{df}{dt}\right)_{i+1/2} \Delta t$$

For solving a ***2D*** problem, e.g., finding the orbit of a satellite in the gravitational field of Earth, we need two differential equations, one for ***x*** and one for ***y***:

DRAFT

$$\frac{d^2x}{dt^2} = \frac{F_x}{m} = -\frac{F}{m}\frac{x}{\sqrt{x^2+y^2}} \quad \text{and} \quad \frac{d^2y}{dt^2} = \frac{F_y}{m} = -\frac{F}{m}\frac{y}{\sqrt{x^2+y^2}}\,.$$

And we know that $F = {}^{G\,m\,M}/_{(x^2+y^2)}$, thus

$$\frac{d^2x}{dt^2} = -\frac{GM}{(x^2+y^2)^{3}/_{2}}\,x = K_x \quad \text{and} \quad \frac{d^2y}{dt^2} = -\frac{GM}{(x^2+y^2)^{3}/_{2}}\,y = K_y$$

This is a pair of coupled differential equations. It is solved as follows:

$$\left(\frac{dx}{dt}\right)_{1/2} \approx \left(\frac{dx}{dt}\right)_0 + K_x\frac{\Delta t}{2} \qquad \left(\frac{dy}{dt}\right)_{1/2} \approx \left(\frac{dy}{dt}\right)_0 + K_y\frac{\Delta t}{2}$$

$$\left(\frac{dx}{dt}\right)_{i+\frac{1}{2}} \approx \left(\frac{dx}{dt}\right)_{i-\frac{1}{2}} + K_x\Delta t \qquad \left(\frac{dy}{dt}\right)_{i+\frac{1}{2}} \approx \left(\frac{dy}{dt}\right)_{i-\frac{1}{2}} + K_y\Delta t$$

$$x_{i+1} \approx x_i + \left(\frac{dx}{dt}\right)_{i+\frac{1}{2}}\Delta t \qquad y_{i+1} \approx y_i + \left(\frac{dy}{dt}\right)_{i+\frac{1}{2}}\Delta t$$

Example:

Despite of the crosstalk (a.k.a. coupling) between ***x*** and ***y***, we can compute the coordinates ***x*** and ***y*** of the satellite as functions of time, letting the *WP43* proceed like we did in the *1D* examples in *OMS 3*:

For ease of handling in a first computation, we set $GM = 1 = a$ and the start values

$$x_0 = 1, \quad \left(\frac{dx}{dt}\right)_0 = 0, \quad y_0 = 0, \quad \left(\frac{dy}{dt}\right)_0 = 1.$$

These 'variable' start values shall be entered in our program using INPUT here:

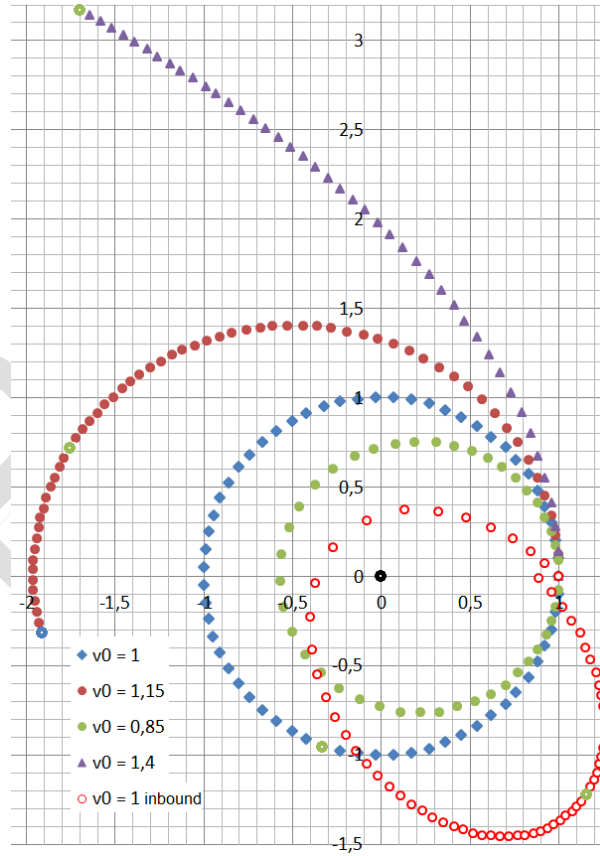
```
LBL 'Satell'
  REM 'Begin of variable initialization'
  INPUT 'x'
  INPUT 'dxbydt'      remember a slash is disallowed in a name
  INPUT 'y'
  INPUT 'dybydt'
  .1                  initialize the remaining fixed start values
  STO 'Δt'
  1                   (for Earth satellites, take GM⊕ from CONST which
  STO 'a'                   would look like # GM⊕ instead of 1 here)
```

DRAFT

```
  0                 start at time zero
  STO 't'
  REM 'End of variable initialization'
LBL 01
  REM 'Begin of the (infinite) time loop'
  RCL 'y'
  RCL 'y'
  x²                y²
  RCL 'x'
  x²
  +                 y²+x²
  -1.5
  y^x               (y²+x²)^-1.5
  RCL× 'a'          a (y²+x²)^-1.5
  ×                 y a (y²+x²)^-1.5 = −Ky
  RCL L             a (y²+x²)^-1.5
  RCL× 'x'          x a (y²+x²)^-1.5 = −Kx.  The stack is [−Kx , −Ky , ...] now.
  RCL 't'
  x>0.?             check time – it will be zero in 1st run
  GTO 02            from 2nd run on go to local label 02
  REM 'The next steps are for the first run only'
  DROP↓             forget t
  2
  /                 −Kx /2
  x⇄y               The stack is [−Ky , −Kx /2 , ...] after x⇄y
  2
  /                 −Ky /2
  x⇄y               The stack is [−Kx /2 , −Ky /2 , ...] after x⇄y
  GTO 03            go to common part of time loop
LBL 02
  REM 'This is used from second run on'
  DROP↓             forget t
LBL 03
  REM 'The common part of the time loop resumes here'
  RCL× 'Δt'         − Kx × Δt  (or half of it in 1st run)
  STO- 'dxbydt'     calculate the new dx/dt
  DROP↓             bring y to the front
  RCL× 'Δt'         − Ky × Δt  (or half of it in 1st run)
  STO- 'dybydt'     calculate the new dy/dt
  RCL 'Δt'
  STO+ 't'          calculate the new time
  RCL× 'dxbydt'     dx/dt × Δt
```

DRAFT

```
  STO+ 'x'          calculate the new x
  VIEW 'x'            and show it for the new point (x,y) to be plotted
  STOP              ----------
  RCL 'Δt'
  RCL× 'dybydt'     dy/dt × Δt
  STO+ 'y'          calculate the new y
  VIEW 'y'            and show it for the new point (x,y) to be plotted
  STOP              -------
  GTO 01
  REM 'End of time loop'
END
```



Plotting the points calculated for these start values will result in a perfect circle as shown by the blue symbols in the diagram – taking 64 *Δt* for one full orbit. We added examples with slightly different start velocities for comparison. The green elliptical orbit takes just 46 time steps, the dark red one 116. Green and blue marks in the other curves highlight the positions after 46 and 64 *Δt* for comparison. For a 40% higher start velocity, violet triangles build a hyperbola.

The innermost red ellipse starts with velocity 1 again but directed 45° inwards ('NW'). Note this curve does not close – why? This happens here since the perigee velocity is too high for the *Δt* chosen. We recommend watching the limits of such numeric models always.

For a smooth descent to a planet or one of the many moons of Jupiter or Saturn, you can introduce a decelerating force even depending on height over ground. Just your imagination is the limit – and your abilities in mathematical modeling.

Solving Differential Equations (2)

Here comes a program solving 1st- and 2nd-order differential equations by the fourth-order *Runge-Kutta* method, translated from *HP-34C Advanced Mathematical Applications*:

A first-order equation is of the form $y' = f(x, y)$, with initial values x_0, y_0, a 2nd-order equation is of the form $y'' = g(x, y, y')$, with initial values x_0, y_0, y'_0.

The solution is a numerical solution which calculates y_n for $x_n = x_0 + n\,s$ (with n = 1, 2, 3, ...), where s is an incremental step specified by the user. The value for s may be changed at any time during the program's execution. This allows solution of the equation for values of x arbitrarily close to a pole ($y = \pm\infty$), or a zero ($y = 0$).

The initial values shall be provided on the stack, e.g., [**s**, y'_0, y_0, x_0, ...]. Results are delivered as sequence of **x**, **y**, and – if applicable – **y'** values.

Equations for 1st order:

$$y_{i+1} = y_i + \frac{1}{6}\,(c_1 + 2c_2 + 2c_3 + c_4)$$

with

$$c_1 = s\ f(x_i\ ,\ y_i)$$

$$c_2 = s\ f\left(x_i + \frac{s}{2}\ ,\ y_i + \frac{c_1}{2}\right)$$

$$c_3 = s\ f\left(x_i + \frac{s}{2}\ ,\ y_i + \frac{c_2}{2}\right)$$

$$c_4 = s\ f(x_i + s\ \ ,\ y_i + c_3)$$

```
LBL 'RKdeq1'
  REM 'Runge-Kutta for 1st order differential equations'
LBL 'Init1'
  STO 00          Begin with [s, y0, x0, ...]
  R↓
  2
  STO/ 00         R00 contains s / 2 now
  R↓
  STO 02          [y0, x0, ...]
  R↓
  STO 01          x0
  STOP
```

DRAFT

```
LBL 'Repet1'
  RCL 02          y
  RCL 01          [x,  y,  …]
  XEQ 'funct1'
  STO 'C1'        (s/2)×f(x, y) = c1/2
  RCL 02
  +               c1/2 + y
  RCL 01          [x,  c1/2 + y,  …]
  RCL 00
  +               [x + s/2,  c1/2 + y,  …]
  XEQ 'funct1'
  STO 'C2C3'      (s/2)×f(x + s/2, c1/2 + y) = c2/2
  RCL 02
  +               c2/2 + y
  RCL 01          x
  RCL 00
  +               [x + s/2,  c2/2 + y,  …]
  XEQ 'funct1'    (s/2)×f(x + s/2, c2/2 + y) = c3/2
  STO+ 'C2C3'     R4 contains  c2/2 + c3/2  now
  ENTER↑          [c3/2,  c3/2,  …]
  +               c3
  RCL 02
  +               c3 + y
  RCL 01          [x,  c3 + y,  …]
  RCL 00
  ENTER↑          [s/2,  s/2,  x,  c3 + y,  …]
  +               [s,  x,  c3 + y,  …]
  STO 's'
  +               [x + s,  c3 + y,  …]
  XEQ 'funct1'    (s/2)×f(x + s, c3 + y) = c4/2
  RCL 'C1'        c1/2
  +               c1/2 + c4/2
  RCL 'C2C3'      [c2/2 + c3/2,  c1/2 + c4/2,  …]
  ENTER↑
  +               [c2 + c3,  c1/2 + c4/2,  …]
  +               [c1/2 + c2 + c3 + c4/2,  …]
  3
  /               (c1 + 2c2 + 2c3 + c4)/6
  STO+ 02         R02 contains  y + (c1 + 2c2 + 2c3 + c4)/6  now
  RCL 's'
  STO+ 01
  RCL 01          s + x = new x
  STOP
```

DRAFT

```
  RCL 02
  STOP
  GTO 'Repet1'
```
new **y**

ad infinitum

```
LBL 'funct1'
  NOP
  RCL 00
  ×
  RTN
END
```
← Store your function here (see example below)

Multiply the function result times **s / 2**

How to proceed:

1. Enter *PEM* and key in the program as listed above
2. Leave *PEM*. Press GTO '**funct1**'
3. Enter *PEM* and key in the function, replacing NOP. Remember ***x*** shall be in x and ***y*** in y at begin.
4. Leave *PEM*. Push the initial values for **s**, **y**, and **x** on the stack. Then call XEQ '**Init1**'
5. Get x_1, y_1, x_2, y_2, x_3, …
   (Feel free to store a new value for **s/2** into **R00** while the program stops.)

Example:

Solve numerically the first-order differential equation

$$y' = \frac{\sin(x) + arctan\left(y/x\right)}{y - ln\left(\sqrt{x^2 + y^2}\right)}$$

where $x_0 = y_0 = 1$. Let **s** = **0.5**. The angular mode must be set to radians.

Keystrokes:

1. Enter *PEM* and key in the program as listed above
2. P/R GTO '**funct1**'
3. P/R ▼ MODE RAD
   STO 05 x≷y STO 06 x≷y →P ln RCL 06 − +/− 1/x
   RCL 06 RCL 05 / TRI arctan RCL 05 sin + x
4. P/R 1 ENTER↑ 1 ENTER↑ 0.5 XEQ PROG Init1
5. 1.50 R/S 2.06 R/S

DRAFT

2.00 R/S **2.78** R/S
2.50 R/S **3.28** R/S ...

Equations for 2nd order:

$$y_{i+1} = y_i + s\left[y'_i + \frac{1}{6}(k_1 + k_2 + k_3)\right]$$

$$y'_{i+1} = y'_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)$$

with $k_1 = s\ g(x_i\,,\ y_i\,,\ y'_i)$

$$k_2 = s\ g\left(x_i + \frac{s}{2}\,,\ y_i + \frac{s}{2}y'_i + \frac{k_1}{8}\,,\ y'_i + \frac{k_1}{2}\right)$$

$$k_3 = s\ g\left(x_i + \frac{s}{2}\,,\ y_i + \frac{s}{2}y'_i + \frac{k_2}{8}\,,\ y'_i + \frac{k_2}{2}\right)$$

$$k_4 = s\ g\left(x_i + s\,,\ y_i + s\,y'_i + \frac{k_3}{2}\,,\ y'_i + k_3\right)$$

LBL 'RKdeq2'	
REM 'Runge-Kutta for 2nd order differential equations'	
LBL 'Init2'	
STO 00	Begin with [**s**, **y'$_0$**, **y$_0$**, **x$_0$**, …]
R↓	
2	
STO/ 00	**R00** contains **s / 2** now
R↓	
STO 03	[**y'$_0$**, **y$_0$**, **x$_0$**, …]
R↓	
STO 02	[**y$_0$**, **x$_0$**, …]
R↓	
STO 01	**x$_0$**
STOP	
LBL 'Repet2'	
RCL 03	**y'**
RCL 02	**y**
RCL 01	[**x**, **y**, **y'**, …]
XEQ 'funct2'	
STO 'K1'	**(s/2)×f(x, y, y')** = **k$_1$/2**
STO I	

DRAFT

```
RCL 00
XEQ A
STO 04
STO I
RCL 00
RCL I
RCL 03
+
RCL 'K1'
XEQ B
STO+ 04
ENTER↑
+
STO I
RCL 00
ENTER↑
+
XEQ A
RCL 04
ENTER↑
+
+
RCL 'K1'
+
3
/
RCL 03
+
STO 03
RCL L
RCL 04
RCL 'K1'
+
3
/
+
RCL 00
ENTER↑
+
STO 'K1'
×
RCL 02
+
```

[**s/2, k_1/2**, …]
Decoding the rest of this program is left to the reader

DRAFT

```
   STO 02
   RCL 'K1'
   STO+ 01
   RCL 01          new x
   STOP
   RCL 02          new y
   STOP
   GTO 'Repet2'    ad infinitum
```

```
LBL A
   RCL I
   RCL 03
   +
   RCL I
LBL B
   2
   /
   RCL 03
   +
   R↑
   ×
   RCL 02
   +
   RCL 01
   R↑
   +
LBL 'funct2'
   NOP             ← Store your function here (see example below)
   RCL 00
   ×               Multiply the function result times s / 2
   RTN
END
```

How to proceed:

1. Enter *PEM* and key in the program as listed above
2. Leave *PEM*. Press GTO '**funct2**'
3. Enter PEM and key in the function, replacing NOP. Remember ***x*** shall be in x, and ***y*** in y, and ***y'*** in z at begin.
4. Leave *PEM*. Push the initial values for **s**, **y'**, **y**, and **x** on the stack. Then call XEQ '**Init2**'
5. Get x_1, y_1, x_2, y_2, x_3, …
   (Feel free to store a new value for **s/2** into **R00** while the program stops.)

DRAFT

Example:

Solve numerically the 2nd-order differential equation $(1 - x^2)\, y'' + x\, y' = x$ where $x_0 = y_0 = y_0' = 0$ and $s = 0.1$.

Rewrite the equation as

$$y'' = \frac{x\,(1 - y')}{1 - x^2}$$

Keystrokes:

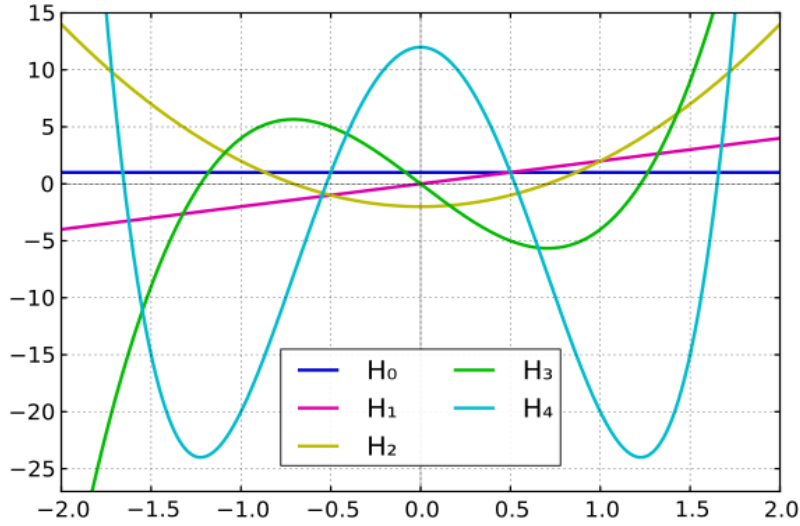
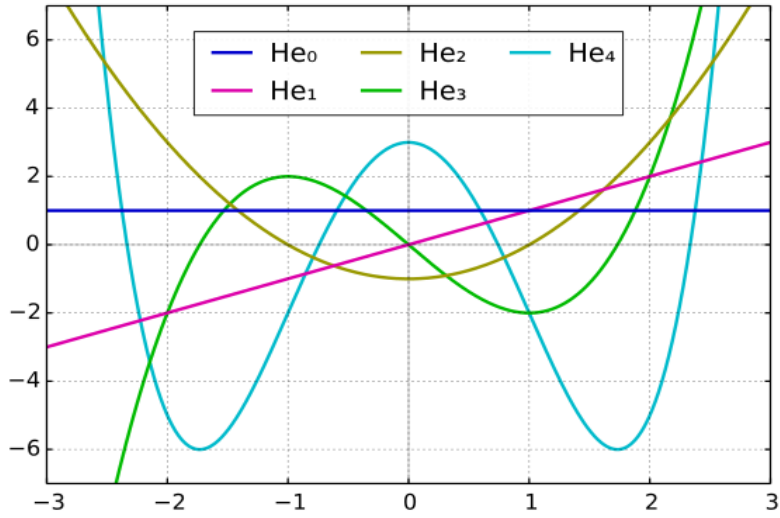
1. Enter *PEM* and key in the program as listed above
2. P/R GTO 'funct2'
3. P/R ▼
   STO 05 R↓ R↓ 1 − RCL 05 × RCL 05 EXP x^2 1 − /
4. P/R 0 FILL 0.1 DISP FIX 4 XEQ PROG Init2
5. 0.100 0 R/S 0.000 2 R/S
   0.200 0 R/S 0.001 3 R/S
   0.300 0 R/S 0.004 6 R/S ...

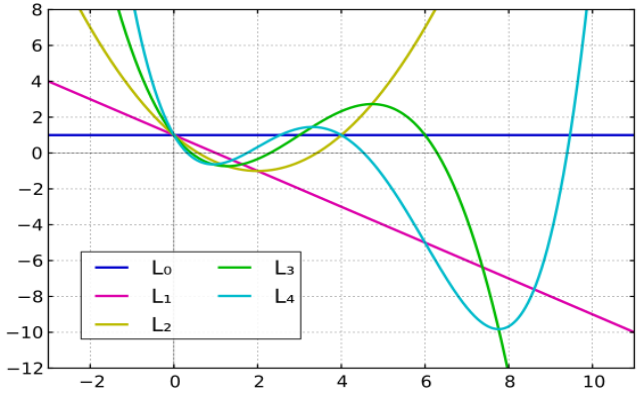
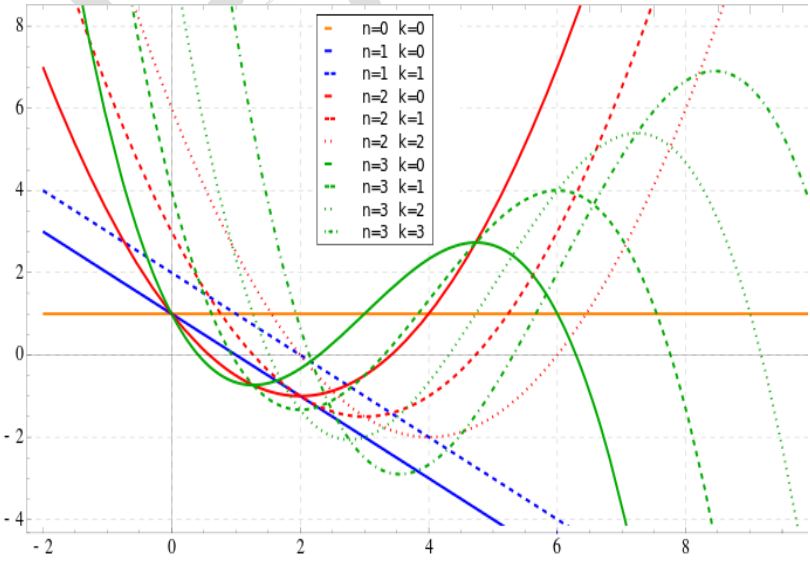
Orthogonal Polynomials

The following seven sets of polynomials are all collected in X.FN'ORTHOG:

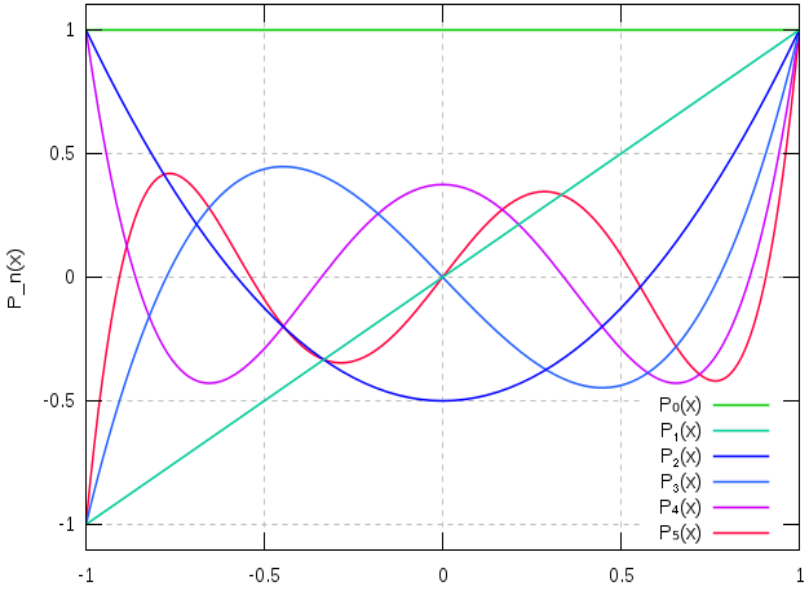
Name	Remarks (cf. pp. 11ff for general information)
H_n	*Hermite polynomials* for probability: $$H_n(x) = (-1)^n\ e^{x^2/2}\ \frac{d^n}{dx^n}\left(e^{-x^2/2}\right)$$ with ***n*** in $\mathbf{Y}$ solve the differential equation $$f''(x) - 2x \cdot f'(x) + 2n \cdot f(x) = 0\ .$$ See the first five polynomials plotted overleaf.

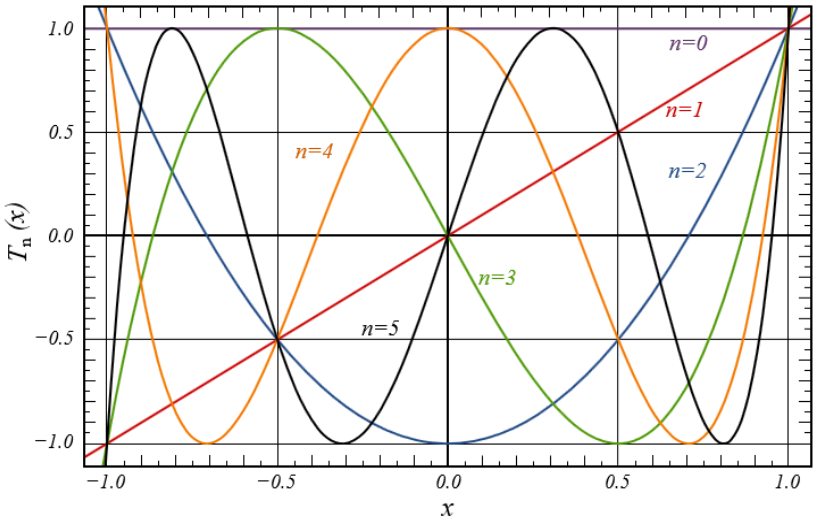
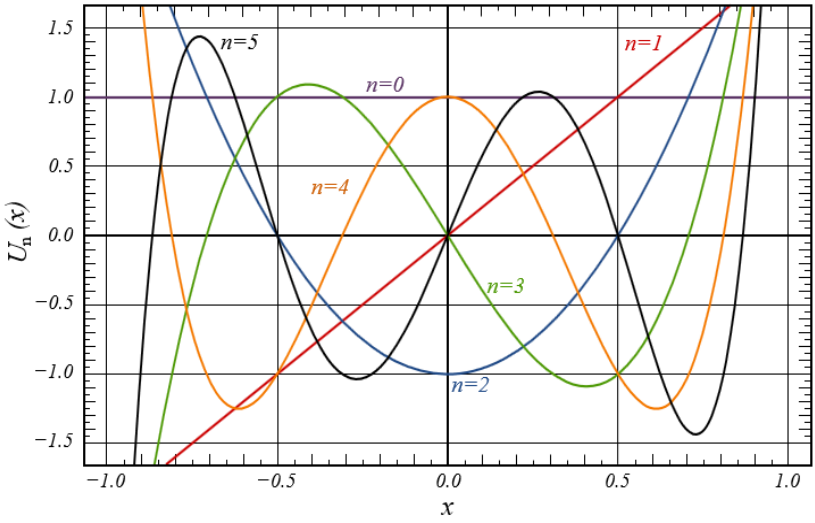
DRAFT

Name	Remarks (cf. pp. 11ff for general information)
	
H_{np}	*Hermite polynomials* for physics: $H_{np}(x) = (-1)^n \; e^{x^2} \; \frac{d^n}{dx^n}\left(e^{-x^2}\right)$ with ***n*** in $\mathbf{Y}$ solve the same differential equation. See the first five polynomials plotted: 

Name	Remarks (cf. pp. 11ff for general information)
L_m	*Laguerre polynomials*: $L_m(x) = \frac{e^x}{m!}\ \frac{d^m}{dx^m}(x^m\, e^{-x}) = L_m^{(0)}(x)$ with ***m*** in $\mathbf{Y}$ solve the differential equation $x \cdot f''(x) + (\alpha + 1 - x) \cdot f'(x) + m \cdot f(x) = 0$ with $\alpha = 0$ (compare $L_{m\alpha}$ below). See the first five *Laguerre polynomials* plotted here. The first three you will find below again. 
$L_{m\alpha}$	*Laguerre's <u>generalized</u> polynomials* (compare L_m): $L_m^{(\alpha)}(x) = \frac{x^{-\alpha}\, e^x}{m!}\ \frac{d^m}{dx^m}(x^{m+\alpha}\, e^{-x})$ with ***m*** in $\mathbf{Y}$ and ***α*** in $\mathbf{Z}$. Some are plotted below (n = ***m***, k = ***α***). 

FDAI

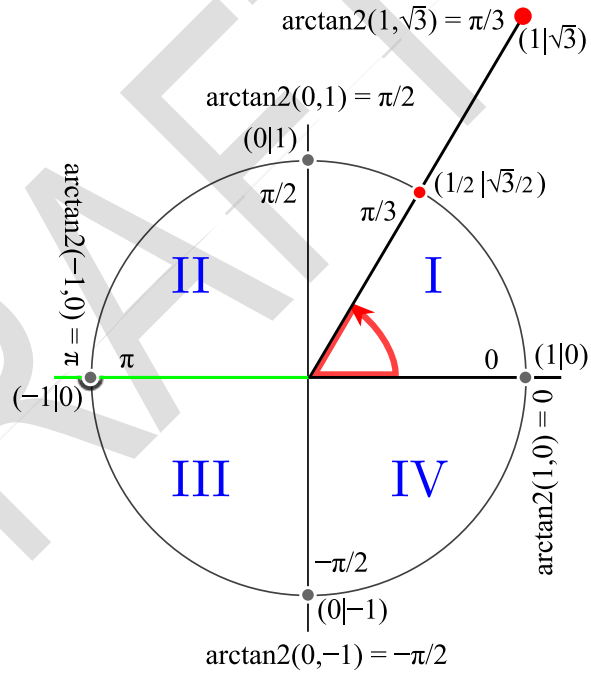
Name	Remarks (cf. pp. 11ff for general information)
P_n	*Legendre polynomials*: $$P_n(x) = \frac{1}{2^n\, n!}\ \frac{d^n}{dx^n}[(x^2-1)^n]$$ with ***n*** in $\mathbf{Y}$ solve the differential equation $$\frac{d}{dx}\left[(1-x^2)\cdot\frac{d}{dx}f(x)\right] + n\,(n-1)\,f(x) = 0\,.$$ The first six polynomials are plotted here: 
T_n	*Čebyšëv* (a.k.a. *Chebyshev, Tschebyschow, Tschebyscheff, originally Чебышёв*) *polynomials of <u>first</u> kind* $$T_n(x) = \begin{cases} \cos(n\arccos(x)) & \text{for } -1 \le \boldsymbol{x} \le 1 \\ \cosh\big(n\,\operatorname{arcosh}(x)\big) & \text{for } \boldsymbol{x} > 1 \\ (-1)^n \cosh\big(n\,\operatorname{arcosh}(-x)\big) & \text{for } \boldsymbol{x} < -1 \end{cases}$$ with ***n*** in $\mathbf{Y}$ solve the differential equation $$f''(x) - \frac{x}{1-x^2}f'(x) + \frac{n^2}{1-x^2}f(x) = 0\,.$$ The plot overleaf shows $T_0(x)$ ... $T_5(x)$.

Name	Remarks (cf. pp. 11ff for general information)
	
U_n	*Čebyšëv polynomials of <u>second</u> kind* $U_n(x)$ with ***n*** in $\mathbf{Y}$ solve the differential equation $$f''(x) - \frac{3x}{1-x^2} f'(x) + \frac{n(n+2)}{1-x^2} f(x) = 0$$ The plot shows $U_0(x)$ ... $U_5(x)$: 

DRAFT

Even More Mathematical Functions

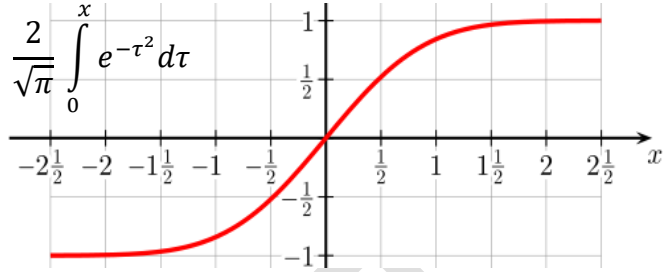
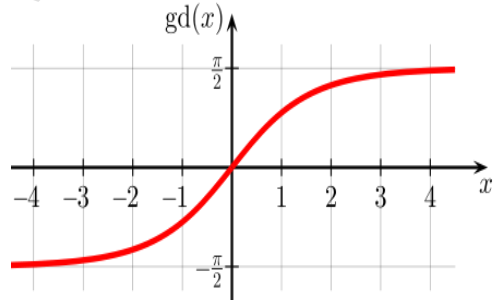
Also all the following functions are found in X.FN. Some of them are for pure mathematics only but were useful at some stages of the *WP 34S* or *WP43* projects, so we made them accessible for the public.

Name	Remarks (cf. pp. 11ff for general information)
AGM	Returns the *arithmetic-geometric mean*. Find more about it here: *http://mathworld.wolfram.com/Arithmetic-GeometricMean.html* .
atan2	Returns the expanded *arcus tangent*. It is explained most easily using this picture showing the returns of arctan2 for five points (x, y): 

DRAFT

Name	Remarks (cf. pp. 11ff for general information)
cn(u,m), dn(u,m), sn(u,m), ψ(u,m), K(m), F(φ,m), E(m), E(φ,m), Π(n,m), Z(φ,m)	$$K(m) = F\left(\frac{\pi}{2}; m\right) = \int_0^{\pi/2} \frac{d\vartheta}{\sqrt{1 - m\,sin^2\vartheta}} \quad \text{with} \quad m \in \mathbb{R}$$ is called the *complete elliptic integral of the 1st kind*. The *incomplete elliptic integral* of this kind is: $$F(\varphi; m) = \int_0^{\varphi} \frac{d\vartheta}{\sqrt{1 - m\,sin^2\vartheta}} = \int_0^{\tan\varphi} \frac{d\omega}{\sqrt{(1+\omega^2)[1+\omega^2(1-m)]}} = u$$ Using u, m, and the 'amplitude' ψ, the following four *elliptic functions* are defined: $$\mathrm{sn}(u; m) = \sin(\psi), \quad \psi(u; m) = arcsin[sn(u; m)],$$ $$\mathrm{cn}(u; m) = \cos(\psi), \text{ and } \mathrm{dn}(u; m) = \sqrt{1 - m\sin^2\psi}$$ The (*in-*) *complete elliptic integral of the 2nd kind* looks like this: $$\left(E(\varphi, m) = \int_0^{\varphi} \sqrt{1 - m\,sin^2\vartheta}\; d\vartheta\right) \quad E(m) = \int_0^{\pi/2} \sqrt{1 - m\,sin^2\vartheta}\; d\vartheta$$ *Jacobi's Zeta* is $Z(\varphi; m) = E(\varphi; m) - F(\varphi; m)\,\frac{E(m)}{K(m)}$. The *complete elliptic integral of the 3rd kind* is $$\Pi(n, m) = \int_0^{\pi/2} \frac{d\vartheta}{(1 - n\,sin^2\vartheta)\sqrt{1 - m\,sin^2\vartheta}}$$ You find more about the *elliptic functions* and *integrals* provided here at https://en.wikipedia.org/wiki/Jacobi_elliptic_functions and https://en.wikipedia.org/wiki/Elliptic_integral (the German texts at https://de.wikipedia.org/wiki/Jacobische_elliptische_Funktion and https://de.wikipedia.org/wiki/Elliptisches_Integral contain a few applications at least). Examples of usage are found, e.g., in https://dlmf.nist.gov/19.30 ff, https://dlmf.nist.gov/22.18 and *...19*; some interesting original texts of *Jacobi* et. al. (of 1849ff) are accessible via http://sites.mathdoc.fr/cgi-bin/rbsm?cc=F__8_h%2a

DRAFT

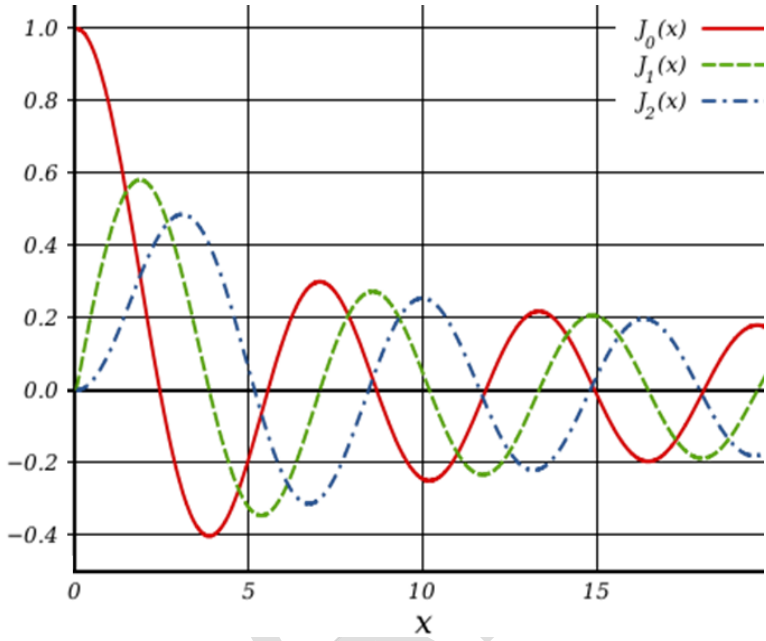
Name	Remarks (cf. pp. 11ff for general information)
erf	… returns the *error function* $erf(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-\tau^2} d\tau$  . Note that $erf\left(x/\sqrt{2}\right) = 2\,\Phi(x) - 1$ where $\Phi(x)$ represents the *standardized normal CDF* as described on p. 275. Beyond statistics, the *error function* may be helpful in heat conduction and diffusion problems, for instance.
erfc	… returns the *complementary error function* $erfc(x) = 1 - erf(x) = \frac{2}{\sqrt{\pi}} \int_x^\infty e^{-\tau^2} d\tau$ This function is related to the *error probability* of the *standardized normal distribution*.
g_d, g_d^{-1}	g_d returns the *Gudermannian function* $g_d(x) = \int_0^x \frac{d\xi}{cosh(\xi)}$ linking hyperbolic and trigonometric functions. See the plot for its *real* values. The *inverse of this function* is  $g_d^{-1}(x) = \int_0^x d\xi / cos(\xi)$ Start reading here for more: *http://en.wikipedia.org/wiki/Gudermannian_function* .

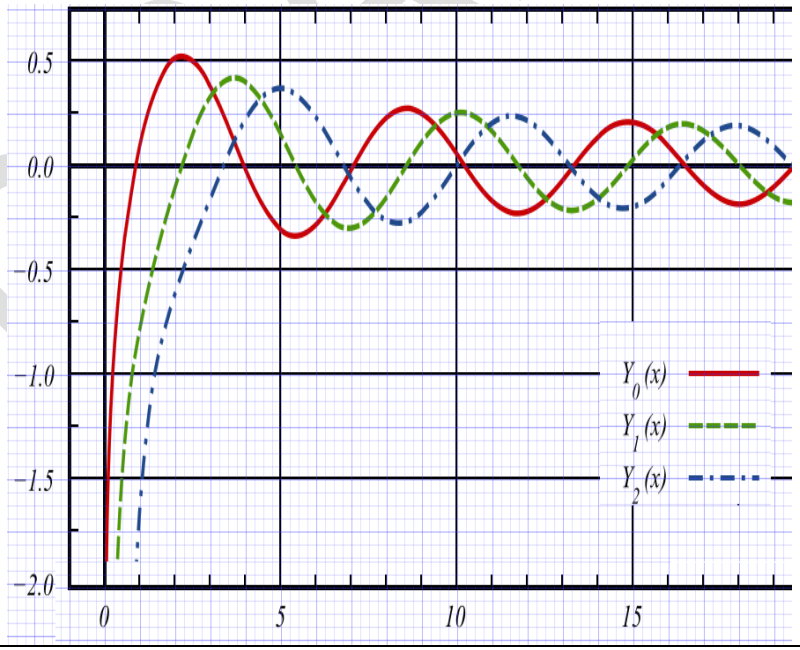
DRAFT

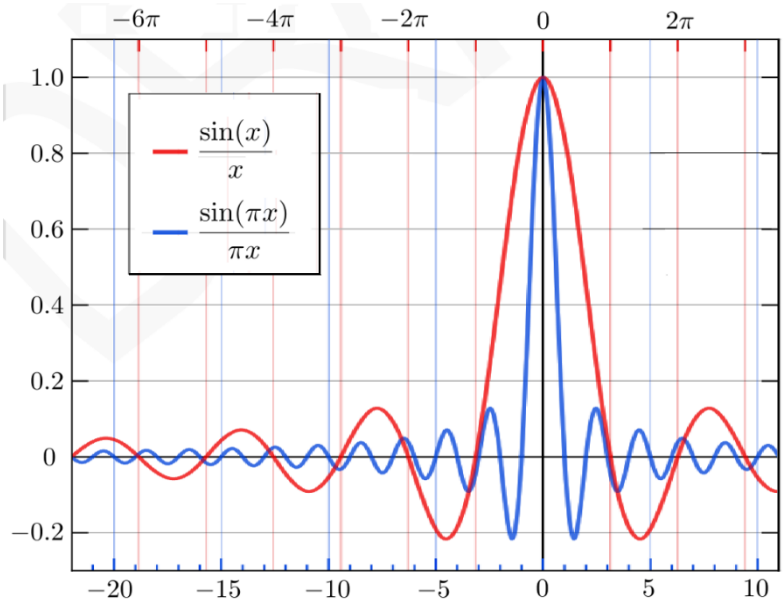
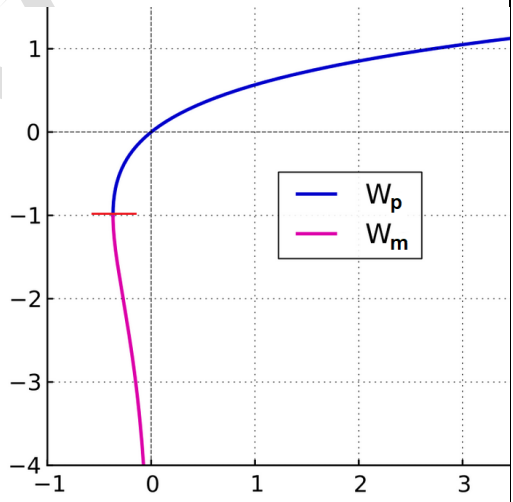
Name	Remarks (cf. pp. 11ff for general information)
I_{xyz}	… returns the *regularized (incomplete) Beta function* $I_{xyz} = \beta_x(x,y,z) \Big/ \beta(y,z)$ with $\beta_x(x,y,z) = \int_0^x \tau^{y-1}\,(1-\tau)^{z-1}\,d\tau$ being the *incomplete Beta function* and $\beta(y,z)$ being *Euler's Beta function* (cf. p. 100 and https://en.wikipedia.org/wiki/Beta_function).
$J_y(x)$, $Y_y(x)$	Generally, the *Bessel functions* solve the differential equation $x^2 f''(x) + x f'(x) + (x^2 - y^2) f(x) = 0$ with $y \in \mathbb{C}$. $J_y(x)$ returns the *Bessel function of first kind* and order y. For arbitrary y, this is $J_y(x) = \sum_{m=0}^{\infty} \frac{(-1)^m}{m!\ \Gamma(m+y+1)} \left(\frac{x}{2}\right)^{2m+y}$ For integer y, this is also $J_y(x) = \frac{1}{\pi} \int_0^{\pi} cos[y\,t - x\,sin(t)]\,dt$ $Y_y(x)$ returns the *Bessel function of second kind* and order y. For arbitrary y, this is $Y_y(x) = \frac{J_y(x)\,cos(y\,\pi) - J_{-y}(x)}{sin(y\,\pi)}$ Watch the different vertical scales of the plots overleaf. Start reading here for more information: http://en.wikipedia.org/wiki/Bessel_function .

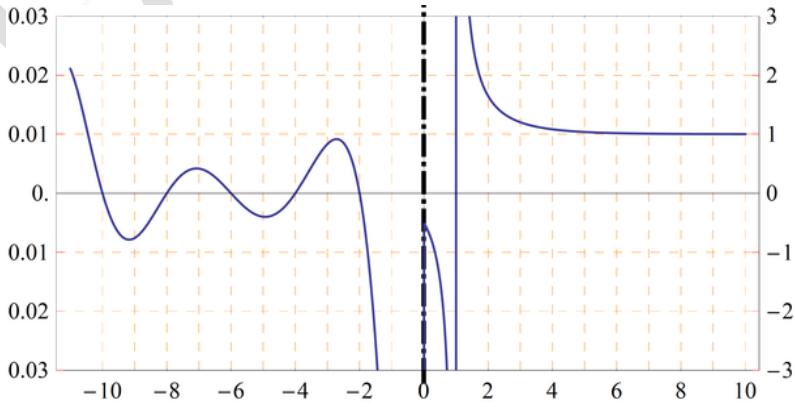
DRAFT

Name	Remarks (cf. pp. 11ff for general information)





Name	Remarks (cf. pp. 11ff for general information)
sinc, sincπ	
W_p , W_m	Return *Lambert's W* with its principal branch (called W_p here) and its negative branch (called W_m for minus). The connecting point is $(x, y) = (-1/e, -1)$. The graph shows the *real* values of both branches.  Start reading here for more information: http://en.wikipedia.org/wiki/Lambert_W_function. Learn more here: http://mathworld.wolfram.com/LambertW-Function.html .

Name	Remarks (cf. pp. 11ff for general information)
Γ_p	Returns the *regularized gamma function* $$P(x,y) = \gamma(x,y) \Big/ \Gamma(x) \quad (see\ \gamma_{XY})$$
Γ_q	Returns the *regularized gamma function* $$Q(x,y) = \Gamma_u(x,y) \Big/ \Gamma(x) \quad (see\ \Gamma_{XY})$$
γ_{XY}	Returns the *lower incomplete gamma function* required for Γ_p above: $$\gamma(x,y) = \int_0^y t^{x-1} e^{-t} dt$$
Γ_{XY}	Returns the *upper incomplete gamma function* required for Γ_q above: $$\Gamma_u(x,y) = \int_y^\infty t^{x-1} e^{-t} dt$$
ζ(x)	Returns *Riemann's Zeta* for *real* $\boldsymbol{x}$, with $$\zeta(x) = \sum_{n=1}^{\infty} \frac{1}{n^x} \quad \text{for } \boldsymbol{x} > 1,$$ and its analytical continuation for $\boldsymbol{x} < 1$: $$\zeta(x) = 2^x\, \pi^{x-1} \sin\left(\frac{\pi}{2}\, x\right) \Gamma(1-x)\ \zeta(1-x).$$ 

Cf. p. 101 for $\Gamma(x)$ and the following for more: https://en.wikipedia.org/wiki/Incomplete_gamma_function

DRAFT

Name	Remarks (cf. pp. 11ff for general information)
	Watch the different vertical scales for the negative and positive x in this plot! And note $\sqrt{6\ \zeta(2)} = \pi$. Look here for more: *http://mathworld.wolfram.com/RiemannZetaFunction.html* .

Note that the *incomplete gamma* and *error functions*, the *Laguerre*, *Legendre*, and *Bessel functions*, as well as some matrix operations were provided on the *Commodore*[175] *M55* (*The Mathematician*, a pocket calculator with 55 keys as pictured below) in 1976/77 already. At the same time, the *S61* (*The Statistician* with 60 keys!) featured the *binomial*, *hypergeometric*, *Poisson*, *chi-square*, *Gaussian*, *Student's t*, and *Fisher's F distributions* as well as the *chi-square test* for fits. Both calculators were accompanied by the *N60* (*The*

[175] This is the company which produced the *C64*, too.

Navigator with 60 keys as well), dedicated to flight navigation. Taking into account that no *HP* pocket calculator ever provided orthogonal polynomials nor the *hypergeometric distribution* until today, *Commodore* offered an impressive early triple.

At the other end of the quality scale, see one of the worst *RPN* implementations next to it. Admitted, it was in the Seventies and didn't cost a fortune (while *HP's* calculators did, in particular beyond the *USA*), but nevertheless precision, power, and memory of *Sir Sinclair's* calculator left a lot of space for improvement while it was definitely the smallest pocket 'scientific' of its time. So it was just nothing more than a relatively cheap replacement of a very good slide rule (which showed comparable precision) before prices of better 'scientific' calculators plummeted.

Beyond what is printed in this appendix, you will find lots of information in the internet about the special functions implemented in your *WP43*. Generally, *Wikipedia* is a good starter – we recommend checking the articles in different languages since they may well contain different material and use different approaches. For applied statistics, the *NIST/SEMATECH e-Handbook of Statistical Methods* (quoted on pp. 269ff) is a competent source. And for pure mathematics, the *NIST Digital Library of Mathematical Functions* and *Mathworld* (both quoted on pp. 316f) may contain more details than you ever wanted to know.

For the functions of *WP43*, we used the *GNU multiple precision arithmetic library* (see *https://gmplib.org/manual/*).

DRAFT

APPENDIX J: INFORMATION FOR ADVANCED USERS

Recursive Programming

Using *local registers* allows for creating a subroutine that calls itself recursively. Each invocation deals with its local data only. Of course, the *RPN stack* is global so be careful not to corrupt it – feel free to use STOSTK for saving it before you call a subroutine and RCLSTK for recovering it thereafter.

Below is a recursive implementation of the factorial. This is an example for demonstration purposes only, since this routine will neither set the *stack* correctly nor will it work at all for input greater than some hundred:

```
LBL 'FACT'
  IP
  x>1.?
  GTO 00
  1
  RTN
LBL 00
  LocR 01
  STO .00
  DEC X
  XEQ 'FACT'
  RCL× .00
  RTN
```

Assume $x = 4$ when you call FACT. Then it will allocate one *local register* ($\mathbf{R.00}$) and store **4** therein. After decrementing x, FACT will call itself.

Then $FACT_2$ will allocate a *local register* ($\mathbf{R.00_2}$) and store **3** therein. After decrementing x, FACT will call itself again.

Then $FACT_3$ will allocate a *local register* ($\mathbf{R.00_3}$) and store **2** therein. After decrementing x, FACT will call itself once more.

Then $FACT_4$ will return to $FACT_3$ with $x = 1$. This x will be multiplied by $r.00_3$ there, returning to $FACT_2$ with $x = 2$. This x will be multiplied by $r.00_2$ there, returning to FACT with $x = 6$, where it will be multiplied by $r.00$ and will finally become **24**.

A Few Words About Factorizing

In a nutshell, FACTOR uses the prime sieve of *Eratosthenes* (some 200 B.C., though the algorithm is much older). Thus, it starts with 2 and continues with 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53, 59, etc. to check if the source number is a factor of the check prime number. This is it – there are no shortcuts, no stored list of prime numbers, nor any other tricks. With a huge source number being a product of two big prime factors, it may take very long to find them (advantage in cryptography).

DRAFT

For quick estimates of the time required for factorizing, the *WP43 Simulator* on my laptop needed ca. 90 minutes to check prime factors up to 10^{9} in 2025-05; i.e. it would need some 900 minutes (15 hours) to check prime factors up to 10^{10} etc. – your computer may be a lot faster, but your calculator is way slower. Feel free to press [EXIT] for aborting FACTOR.

A Few Words About the Solver

In a nutshell, the *Solver* takes the start values for the unknown for calculating the function to be solved, and proceeds as outlined in the *HP-42S OM* on pp. 186f.

In most situations, the calculated root is an accurate estimate of the theoretical, infinitely precise root of the function. An ideal solution is one for which *f(x)* exactly equals 0. However, a nonzero value for *f(x)* is often also acceptable, because it results from approximating the root with limited (…) precision.[176]

While solving, as soon as SOLVE has found two results $f(x_1)$ and $f(x_2)$ of opposite signs, it divides the interval $|x_2 - x_1|$ in two halves and looks in which of them the sign change occurs. You can observe this process 'live' on the screen of your *WP43*:

SOLVE will stop when $|f(x_{root})| \leq u$ with *u* being the least displayable number within the screen format selected.[177] Thus, for SCI 15 solving may end with a solution like:

With 34 significant digits, the difference between the last two approximations to the location of the root turns out being 5×10^{-33} here.

[176] Quoted from *HP-42S RPN Scientific PET*, p. 108.

[177] This applies if SOLVE returns claiming '*root found*'. It does not for poles, constant curve shapes, local or asymptotic extrema (cf. *HP-42S* manuals).

Feel free to look up *Sect. 12* of the *HP-42S OM*. See the *HP-34C OH and Programming Guide* (*Sect. 8* and *App. A*) or the *HP-15C OH* (*Sect. 13* and *App. D*) for more information and some caveats.[178] Additionally, finding roots at the limits of function domain may be a challenge.

More Than a Few Words About the Integrator

When you know the analytical result of an integral, you should use this knowledge. When an integral turns out being difficult to evaluate analytically, however, it may be easier solving it numerically.

Remember 〔∫〕 and its siblings return the approximation for the integral in **X** and an upper bound for its uncertainty in **Y**.

We recommend looking at the *HP-34C OH*, *Sect. 9* and *App. B*, of 1979 when the *Integrator* was first featured in a pocket calculator.[179] The *HP-15C CE OH*, *Sect. 14* and *App. E*, and in particular the *HP-15C CE Advanced Functions Handbook*, *Sect. 2*, pp. 45 - 68, of 2023 may be an even better read, also about difficult integrals. This topic is also covered in the *HP-42S OM, Sect. 13*, and its *PET*, *Sect. 4*. Note, however, that *HP*'s integration algorithm differs from ours.

WP43 uses the *tanh-sinh quadrature* integration algorithm published by *Michalski* and *Mosig,* actually its version written for the *WP 34S* by *César Rodríguez*. See *https://newtonexcelbach.com/2020/10/29/numerical-integration-with-tanh-sinh-quadrature-v-5-0/* for more. This is a very capable algorithm. Nevertheless, functions containing poles, narrow peaks or drops, and/or rapid oscillations in a wider range within the integration interval may carry it beyond its limitations.

We recommend checking and analysing the integrand and applying suitable methods, e.g., subdividing the integration interval or transforming variables.

Example 1:

$$\int_0^{1000} e^{\sin(x)*\cos(x)}\,dx = ?$$

[178] Solving as described above was introduced with the *HP-34C* by *William Kahan* consulting *HP*.

[179] See *Handheld Calculator Evaluates Integrals* by *William Kahan* in the *HP Journal* of 1980-08 (*http://hparchive.com/Journals/HPJ-1980-08.pdf*), pp. 23 - 32. A highly recommended introduction from a mathematician!

Solution (with ACC = 10^{-33}):

a) Let your *WP43* integrate straight ahead. The result will be 1 038.923 for the integral with y = 0.016.

b) Note this integrand is oscillating a lot – but luckily it is periodic. Thus,

i. Let your *WP43* compute one period $\int_0^{2\pi} e^{\sin(x)*\cos(x)}\,dx$. It will return 6.682… with y = 1.47…×10^{-35}.

ii. Divide the integration interval by 2π. This returns 159.15… Hence multiply the result **i** times 159.

iii. Let your *WP43* compute the integral remaining at the upper end of the integration interval $\int_{318\pi}^{1000} e^{\sin(x)*\cos(x)}\,dx$. It will return 1.398… with y = 8.91…×10^{-34}.

iv. Add the results **ii** and **iii**, returning

1063.846 894 403 590 671 456 757 649 301 007

with a total y = 3.24×10^{-33}.

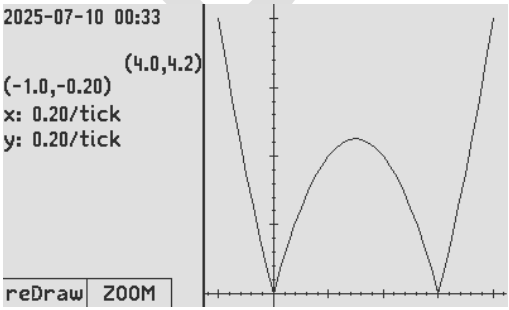
Note that the result **a)** deviates far more from the truth than the estimated error returned in y there.

Example 2:

$$\int_{-15}^{+15} |(x-3)x|\,dx = ?$$

Solution (with ACC = 10^{-33}):

a) Let your *WP43* integrate straight ahead. The result will be 2 258.993 38… with y = 1.4×10^{-5} (the crucial interval [0; 3] is not seen automatically).



b) DRAW the integrand (the picture here shows it between -1 and +4, drawing it between -15 and +15 its bump is hardly seen). It has two roots, 0 and 3. Due to the |…| function, the integrand has sharp corners at these locations – many automatic algorithms do not like

DRAFT

sharp corners. So, let's split the integral in three intervals:

i. $\int_{-15}^{0} |(x-3)x|\,dx = 1462.5$ with $y = 3.6 \times 10^{-35}$

ii. $\int_{0}^{3} |(x-3)x|\,dx = 4.5$ with $y = 1.4 \times 10^{-35}$

iii. $\int_{3}^{15} |(x-3)x|\,dx = 792.$ with $y = 5.1 \times 10^{-35}$

iv. Add the results **i** – **iii**, returning 2259. ± 6.4×10^{-35}, i.e. **2259** exactly.

c) This integral is easily evaluated analytically, on the other hand:

$$\int (x^2 - 3x)\,dx = \frac{x^3}{3} - \frac{3}{2}x^2$$

i. $\Rightarrow \int_{-15}^{0} \ldots = -\left[\frac{1}{3}(-15)^3 - \frac{3}{2}(-15)^2\right] = 1462.5$

ii. $\int_{0}^{3} \ldots = \frac{1}{3}3^3 - \frac{3}{2}3^2 = -4.5$

iii. $\int_{3}^{15} \ldots = \frac{1}{3}15^3 - \frac{3}{2}(15)^2 - (-4.5) = 792$

iv. $\Rightarrow \int_{-15}^{15} |(\ldots)|\,dx = 1462.5 + 4.5 + 792 = \mathbf{2259}$

So we know the numeric *Integrator* works correctly here for the three intervals. Note it was deceived, however, by the 'pathetic' function when trying to integrate in one step – again, the result **a)** deviates far more from the truth **c)** than the estimated error returned in y there. Beware!

There are seven finite integrals recommended by *Valentin Albillo* (Spain) just for testing the capabilities of numeric integration algorithms. These read:

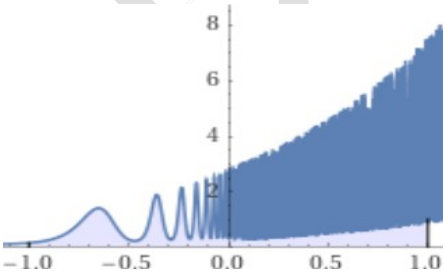
Integral	*Valentin's* result / *WP43* result with ACC = 10^{-33}
I1. $\int_{0}^{1} \cos(x)\ln(x)\,dx$	−0.946 083 070 367 183 014 941 353 313 823 1… **−0.946 083 070 367 183 014 941 353 313 823 1**79… with y = 3×10^{-35}
I2. $\int_{0}^{1} \left(1/\sqrt{x}\right)dx$	2 2. exactly, with y = 1×10^{-35}

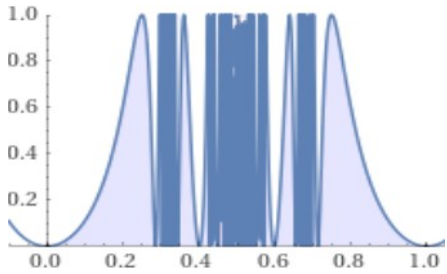
DRAFT

Integral	*Valentin's* result / *WP43* result with ACC = 10^{-33}
I3. $\int_0^1 ln\left(\Gamma(x)\right) dx$	$ln\left(\sqrt{2\pi}\right)$ = 0.918 938 533 204 672 741 780 329 736 405 617 5 **0.918 938 533 204 672 741 780 329 736 405 617** 6 with $y = 6\times10^{-35}$
I4. $\int_0^1 \left(1/\sqrt{-ln(x)}\right) dx$	$\sqrt{\pi}$ = 1.772 453 850 905 516 027 298 167 4… **1.772 453 850 905 516 013** 219 … with $y = 3\times10^{-19}$
I5. $\int_0^1 x^{-0.99}\, dx$	100 **99.998 345 6**60… with $y = 7.1\times10^{-7}$ [180]
I6. $\int_{-1}^1 e^{\,x+sin\left(e^{e^{e^{x+\frac{1}{3}}}}\right)}$	2.996 339 811 976 535 379 182 861 556 235 501 887 668… **2.975**… with $y = 0.017$ [180]
I7. $\int_0^1 sin^2\{tan[tan(\pi x)]\}$	$\left[1 - e^{-2\ tanh(1)}\right]/2$ = 0.390 992 162 151 53… **0.398**… with $y = 0.06$

Computing these integrals on your *WP43* may take some time – *USB* power supply is recommended; then you will be done in less than two minutes. Above results are just for your information. Note **I5** is easily solved analytically but its numeric computation is hampered by the pole at ***x*** = 0.

Look at the plots of the last two integrands below and you see why computing I6 and I7 is so difficult – both are frequently oscillating but not periodic in a useful way. Luckily, integrands like these are far from usual in real-world problems.





[180] Note the true value is not covered by y here.

At the bottom line, we recommend taking good care by analysing each integrand before throwing it to the *Integrator*. It may be deceived easily by particular 'pathetic' features of the integrand. No automatic on earth can solve any integral for you correctly – you have to check in advance and set up the problem properly. Draw what can be drawn, your *WP43* will support you. Circumvent possible pitfalls by applying suitable methods.

And check automatic results for their plausibility as well, e.g., by dividing intervals. Like other automatisms provided on your *WP43*, also numeric integration will be useful and beneficial based on your solid understanding and experience only.

Differentiating Real Functions Using Complex Numbers

The complex-step derivative formula[181] tells $f'(x) \approx {Im[f(x + i\,h)]}/{h}$ for a small ***h***. Thus, we can use this method for differentiating and looking for extrema $(f'(x) = 0)$ of a function, for **example**,

$$f(x) = [sin(x)]^{e^x}$$

```
LBL 'dF'
   MVAR 'x'
   MVAR 'h'
   RCL 'x'
   RCL 'h'
   RE→CX      compose x + i h.
   XEQ 'fx'   please turn overleaf for fx
   CX→RE      decompose the result, returning the imaginary part in x.
END
```

```
LBL 'fx'
   REM 'All functions within this routine operate on complex numbers.'
   sin
   RCL L
   eˣ
   yˣ
END
```

Starting with *SDC*, enter

MODE RAD

[181] See prerequisites in https://en.wikipedia.org/wiki/Numerical_differentiation#Complex-variable_methods. Thanks to *Thomas Klemm* for pointing me to this.

DISP FIX 5
ADV SOLVE PROG ('dF')
E 6 +/- h 14 x 14.25 x x returns x= 14.137 17 for the location of the extremum in this interval.

Executing 'fx' three times you will find out if that is a maximum or minimum.

A Few Words About Random Numbers

No fixed algorithm can generate true random numbers. What can be created are pseudo random numbers (*PRN*), i.e. typically *real numbers* equally distributed in a fix range which look like random. Once started, the sequence of these numbers will only repeat after a very long interval, meaning after very many calls of the *PRN* generator.

The algorithm for RAN# on your WP43 starts either

1. from *SDC* (e.g., after RESET) or
2. with an initial 'seed' number $\boldsymbol{s} \in [0; 1)$ you enter explicitly (***s*** SEED) or
3. with a seed taken from the calculator clock at execution time of **0** SEED.

For equal seed numbers, the following sequence of *PRN* is fully determined.. So you will get repeatable sequences of *PRN* for cases 1 and 2 above which is advantageous for debugging. If you want to get more random sequences of *PRN*, however, choose option 3.

A Few Words About Uncertainty, Significance, and Confidence

There are two ways to look at this:

1. **Multiples of σ (sigma) are the agreed way of indicating uncertainties in science.** 1 σ is generally used from physics to psychology, assuming the underlying data are distributed normally (a.k.a. *Gaussian*).[182] Yes, this corresponds to only 68% confidence, but as long as you know 1 σ, you can determine 2 or 3 σ most easily if you want or need them, corresponding to some 95% or 99% confidence. Physicists at *CERN* require 5 σ or even more before publishing the discovery of a new particle, but that's extreme.

[182] This assumption should be checked and verified – but this is 'forgotten' frequently. Then, however, the results miss a solid foundation at least or may be even simply wrong.

2. **95%, 99%, and 99.9% are common confidence limits or significance levels in statistics**, also applied to testing of hypotheses. Which level is called *significant* may depend on country, discipline, or industry. The examples in *HP's* manuals took 95%, so that seems to be (or have been) the standard in the *USA*. Car industry was also known to work with 95%. Looking at *DQS* (the leading German quality institution, known in the *USA* as well), 99% are the level for significance, and 99.9% for high significance. The remainder to 100%, i.e. 5%, 1%, or 0,1%, are the respective error probabilities. Cf. examples in *OMS 2*.

A Few Words About Curve Fitting

The world is full of jokes about curve fitting since there is a whole lot of mistakes one can make (cf. *App. I*) – one of the most popular is taking more parameters than the underlying data points support. There was a saying: *Using four parameters, you can fit an elephant*;[183] but, of course, the underlying hypotheses or theories determine the number of parameters needed – just don't use more.

Fitting makes most sense based on many data points. If these are acquired experimentally, their errors should be known, usually plotted as error bars – exceeding the sensible capabilities of *WP43*. Nevertheless note that if your error bars represent $\pm s_m$ (as usual) then a 'good' fit curve will go through $^2/_3$ of all your error bars 'only' (cf. previous chapter), and the other bars should scatter symmetrically around the curve. If your model curve hits significantly more than $^2/_3$ then it is either too complex for your data or you overestimated the experimental errors – if it hits significantly less then it is either too simple or you underestimated the errors. A fit curve going through all your many data points is the opposite of optimum!

A Few Words About *TVM*

In business life, the *HP-12C* is the gold standard for computing financial transactions, being on the market for over 40 years so far. The *TVM* user interface of that dedicated financial calculator slightly deviates from the one of your *WP43* since we wanted to spell out the meanings of those equation parameters we found ambiguous. See the following **example** of an annuity due (20 000 US$ for four

[183] Look at the possible curves – this saying makes more sense than you might have thought.

years and 3%/a, payments at the end of each month):

HP-12C	*WP43*	Remarks
f 2	DISP FIX 2	Standard for financial problems
	12 per/a	Almost standard
CLEAR FIN		Clearing is dispensable on *WP43*
END	FIN End	
48 n	48 n_{PER}	Total number of payment periods
3 12÷	3 i%/a	In offers, the annual interest rate is generally specified. The *HP-12C*, however, needs the interest per payment period (do not press i in this example!).
2 EEX 4 PV	2 E 4 PV	
0 FV	0 FV	
PMT returns -442.69	PMT returns PMT = -442.69	*TVM* automatically solves for the variable being called without any numerical input heading.
17 keystrokes	16 keystrokes	

Generally, the *TVM* equation

$$0 = PV + (1 + i\,p)\,\frac{PMT}{i}\,[1 - (1+i)^{-n_{PER}}] + FV\,(1+i)^{-n_{PER}}$$

(cf. *OMS 5)* is solved for one of its variables (the unknown) based on knowing all its other parameters. *TVM* actually uses the *Solver* for doing this. Note all *TVM* parameters are kept by your *WP43* until they are overwritten either by input or solving.

Translation of *WP43* Characters Deviating from *UTF-8*

For coding and writing *WP43* programs in readable text files, `.rtf` files could be used, just containing one line of text per program step. Each line may encompass a mix of

- plain text,
- *italic* text,

DRAFT

- superscripts and subscripts, optionally *italic*,

  and shall be closed by CR and LF.

Any *rich text* editor will do (e.g., *WordPad*) as long as all *WP43 characters* are printable using common fonts. This holds for all of them complying with *UTF-8.* For the *characters* deviating, the following table lists alternative *characters* (altern.) to be used for entering these:[184]

WP	*UTF*	altern.		*WP*	*UTF*	altern.		*WP*	*UTF*	altern.
$\frac{o}{c}$	00a9							$\overline{y}$	0232	See below
$\overline{x}$	0378	*m* (mean)		$\hat{x}$	0379	*f* (forecast)		$\overline{x}$	037f	
|	2027			$^{-1}$	2072	/		E	2073	E [185]
*	208f	*		∞	209e	*00*		∞	209f	*00* [186]
e	2147	Call from CONST		*i*	2148	Call from CONST				
0	2160	0	...	9	2169	9		+	216a	+
-	216b	−		→	21c0	–> [187]				
Δ	21c9			∇	21cb			≡	21cc	
↶	21cd			∎	220e					

[184] Some reasoning is given in parentheses. – A few *WP43 characters* required for spelling *items* appear in dual combinations only, and for some of them the alternatives are just dual combinations as well.

The colors emphasize superscript and subscript letters. *WP43 characters* printed in light grey fields appear in *menus* only (neither in command *names* nor in constants), those in light blue fields are just used on the keyboard. Those printed in grey fields are exclusively for system output, those in dark grey fields are designed but not used presently yet.

[185] This is for entering exponents. To be valid, either a digit or a blank must precede this **E**, and it shall be directly followed by either a digit or a minus sign (**-**) and a digit.

[186] Feel free to type italic ***00*** for the regular *character* ∞.

[187] This alternative can be used for entering a common right arrow as well – it will be translated in the arrow suiting the target.

WP	UTF	altern.		WP	UTF	altern.		WP	UTF	altern.
√x	221a	2r (∛)		∛x	221b	3r (∛)		∛y	221c	xr (∛)
⊕	2295	E (Earth)								
α	2296	α or a		δ	2297	δ or d		μ	2298	μ or m
⊙	229a	S (Sun)			229b					
⌊	2421			⁄	2422					
⁄	2425			⁄	2426			※	2427	
⁝	2428			⁝	2429				242a	b (battery)
⊼	242b			U	242c	U				
UK	242d	k		US	242e	s			242f	!
⚠	2430	p		⚠	2431	See left				
⚠	2432	q		⚠	2433	See left		ψ	2434	
1	2460	1	…	9	2468	9				
10	2469		…	16	246f			10	2470	
a	2482	a	…	z	249b	z				
a	249c	a	…	z	24b5	z				
A	24b6	A	…	Z	24cf	Z				
A	24d0	A	…	Z	24e9	Z				

The following *characters* are not allowed in any file names:

blank space, ", &, *, comma, point, /, :, <, >, ?, \, and |.

The following *characters* must not appear in any user-defined *variables*, *global labels*, or *menu names* (cf. p. 332):

blank space, !, (,), +, comma, −, /, :, ;, =, ^, |, and ×.

((Italic digits and letters used so far: *0, 2, 3,* a, *b,* d, E, *f,* k, *m,* m, *p, q, r,* S, s, *U, x,* and *!*.))

Where to Find Programs Readily Available?

At *https://technical.swissmicros.com/dm42/fat/PROGRAMS/*, *SM* offers a collection of programs for their *DM42*. Although *WP43* is not code compatible to *DM42*, these routines may be helpful since the keystroke programming paradigm is the same.

Furthermore, there are sections in both *https://forum.swissmicros.com/index.php* and *https://www.hpmuseum.org/forum/index.php* where user programs for the keystroke programmables of *HP*, *DM*, and *WP* calculators can be found, copied, and adapted, if applicable.

DRAFT

APPENDIX K: THE MOST HONORABLE MEMBERS OF THE PILOT RUN

We know of 30 pilot samples distributed in 2022-10 and the weeks thereafter. *SM* tells the pilot run delivered 35 *WP43* calculators. The following serial numbers are owned by people communicating on the *SM* forum sometimes:

00000	*Michael, Walter*	00025	*toml_12953*
00001		00026	*gmac42*
00002	*DA74254*	00027	
00003	*(Walter)*	00028	
00004		00029	*Mihail*
00005	*Over_score*	00030	*JanneF*
00006		00031	*Mihail*
00007	*Over_score*	00032	*ctrclkws*
00008		00033	*sca*
00009	*Thomas_ER*	00034	
00010	*tuxfish*	00035	*srk00*
00012	*RJvM*	00036	
00013	*rprosperi*	00037	
00014	*rprosperi*	00038	
00015	*Jaymos*	00039	
00016	*Dany R.*	00040	*HP-Collection*
00017		00041	
00018		00042	*dlachieze*
00019		00043	
00020		00044	*Patrick*
00021	*pauli*	00045	
00022	*pcscote*		
00023	*HP-Collection*	000xx	*akaTB*
00024	*cdmackay*	000xx	*ben.titmus*

DRAFT

APPENDIX L: MAKING A WP43 STAND FOR LESS THAN $2 COST

Your WP43 is conveniently carried in your pocket and comfortably held in your hand. Operating it on your desk, depending on actual lighting conditions, you may want it a bit more inclined.

The stand pictured here was made starting with a bamboo smartphone stand distributed by a big Swedish furniture shop (offered for less than 2 US$ in 2022, less than 2 € in 2025) and a flat file from my toolbox. Optionally glue or stick three or four little rubber snippets or the like on its bottom to keep it from sliding on slippery surfaces, drill a decent 1 mm hole for the inevitable reset pin, and you are done.

APPENDIX M: RELEASE NOTES

Version	Date	Release notes
0	29.11.12	**Official project start with first publication of the *43S* concept and a layout** on one of the forums of the *Museum of HP Calculators* (*https://www.hpmuseum.org/cgi-sys/cgiwrap/hpmuseum/archv021.cgi?read=234685#234685*). Though internal discussions started in September 2011, and there are even far older traces of a '*43S*' denoting a '*Super HP-42S*', though in various more or less fictional cases – pure vapourware ™.
0.1	2.2.14 23.5.15	Manual set up based on the one of *WP 34S*. Passed to *Jake Schwartz*, *Eric Smith*, and *Richard Ottosen* for first information. See the vivid discussions starting in February 2014 in this thread *https://www.hpmuseum.org/forum/thread-593.html* (it was stopped at 10.1.16 but still is among the three threads most viewed on that forum after over nine years).
0.2	3.10.15	Update based on *Jake's* feedback and further thoughts, distributed to *Eric*, *Jake*, *Marcus v. Cube*, and *Paul Dale.*
0.3	21.3.16	Split the manual in three; moved LBL onto the keyboard, renamed STOM to STOCFG, RCLM to RCLCFG, SERR to s_m, and SERR$_w$ to s_{mw}; refined the *Key Response Table*. Passed to *Michael Steinmann* for information.
0.4	28.3.16	Renamed LOGS to EXP and [EEX] to [E]. Added hardware information from 2nd manufacturer.
0.5	29.10.16	Returned [EEX]. Changed keyboard layout.
0.6	22.8.17	Merged the Applications and Owner's Manual. Changed the input order of *complex number* parts on Paul's request. Changed keyboard layout introducing D.MS, SST, BST, and % while removing ŷ, RAN#, FRC, and CFIT. Put CFIT into STAT and FRC into MODE. Placed OFF below EXIT for easier customizing. Renamed cc to C5, [EEX] to [E], STOPW to TIMER, SHOW to REGS, SOLVE to ADV, DLINES to DSTACK, 12h to CLK12, and 24h to CLK24. Replaced IND by →. Deleted %MG since covered by Δ%, added EIGVAL and EIGVEC. Swapped CNST and CONST. Defined the echo rows for alphanumeric and command input. Expanded and modified the *character* sets for better use of display space. Added the *QRG*.

DRAFT

Version	Date	Release notes
0.7	2.4.18	Changed keyboard layout. Replaced the labels BST by ▲, SST by ▼, and UNDO by ↶; added some *alpha input mode* reminders on the keyboard. Added AGRAPH, CLLCD, EQ.xxx, HYP, J/G, M.GOTO, ORTHOF, PIXEL, POINT, TDISP, and 🖶USER. Moved the background considerations out of *ReM App. D*. Introduced **K** as *alpha register* for alphanumeric constants in programs. Removed *fraction data type*. Extended *items* from 6 to 7 *characters* to match *HP-42S*. Specified *DTs* more precisely in *ReM App. D*. Reduced the maximum number of *local registers* from 888 to 100. Deleted JG1582 and JG1752. Renamed two commands for TVM. Put SUMS in STAT. Renamed the trigonometric and hyperbolic functions according to mathematical standards, and 🖶CHR to 🖶CHAR. Redistributed the chapter about constants. Modified STATUS display. Refined the unit conversions to ensure *SI* on one side. Specified 0 SEED. Expanded *App. A*. Added formula output for L.R. Modified CPX?, DBL?, and REAL?. Changed output of binary tests for compatibility with *HP-42S*.
0.8	7.5.18 20.9.18	Changed keyboard layout: introduced TRG containing trigonometric functions, removed HYP into EXP and π to g-shifted /, swapped some shifted labels. Refined the chapters about register arithmetic, *Command Parameter Input*, *Alphanumeric Input*, *Matrix Calculations*, and *Orthogonal Polynomials*. Introduced CLCVAR and more vintage examples. Rearranged *temporary information* on the screen. Renamed REGS to RBR and CLx to CLX. Deleted ANGLE. Corrected errors and inconsistencies. Added one more example. Moved the key response table into an appendix.
0.9	3.1.19	Martin set up the project in GitLab in 2018-12. Removed *angle DT*. Added another industrial application and many more examples. Exchanged keyboard pictures due to changed bezel. Expanded *App. B*. Added SHOW for displaying full precision of *DP* numbers and FBR for browsing our two fonts. Added the overlay drawing. Modified functionalities of EXIT and 1/x to match *HP-42S*. Added a chapter about curve fitting. Modified functionalities of ENTER↑ and ↶. Expanded *App. K*. Renamed DOUBLE to →DP. Added →SP and conversions of *quarts*. Rearranged X.FN. Replaced USR by UM. Changed keyboard moving UM, √x, and TRI. Moved 🖶 to f R/S. Added XIN and XOUT. Added a chapter in *App. E* and information about infinite integers Extended the domain of GCD and LCM. Refined and corrected.

DRAFT

Version	Date	Release notes
0.10	3.3.19	Returned *angle DT* and αSR. Added IDIVR and VANGLE. Refined FP, IP, IMPFRC, PROFRC, SDIGS?, →DP, →HR, →INT, →REAL, →SP, explanation of ALL, the summary of integer functions, and handling of long alpha strings. Modified contents of CPX, MATX, and α●. Added a summary of matrix functions. Removed the ON-key combinations. Modified MEM?. Rewrote the angular conversions. Renamed infinite and finite integers to *long* and *short integers*. Added a chapter about ±∞ and NaN. Modified RBR and the menu for STO and RCL. Removed 🖶 from the keyboard. Renamed X_u to X_e for the distributions.
0.11	8.5.19	Changed keyboard making CC primary and user mode shifted, removing x^2, $x\sqrt{}$, and DSP, adding |x|, DROP, and SHOW, and moving some shifted labels. Modified BITS, CLREGS, CNST, CPX, DISP, EXP, INTS, MODE, PARTS, SHOW, STAT, U→, αMATH, the division matrix, *DT* conversions, and the *Quick Reference Guide*. Added conversions of *barrels*, *carats*, and *fathoms*. Deleted DSP. – Separated predefined *variables*. Refined *Sect. 6*. Added $\bar{x}_H$, $\bar{x}_{RMS}$, nine statistical sums and five curve fit models. Split STAT in STAT and SUMS; renamed RMDR to RMD, L_n to L_m, $L_{n\alpha}$ to $L_{m\alpha}$, Π to Π_n, Σ to Σ_n, and some constants to avoid search ambiguities. Refined *App. J*, *Sect. 3 and 4*, →INT, CLR, and the functions of ▲ and ▼. Put SUMS instead of RMD on the keyboard, moved ADV, BITS, CATALOG, EQN, FILL, INTS, MATX, MODE, PROB, RTN, SHOW, STAT, and α.FN. Rearranged A…Ω and *Sect. 2* of the *OM*.
0.12	16.10.19	Rearranged the appendices of the *ReM* from *App. D* on. Expanded *App. A* of the *OM* and *App. K*. Deleted the standardized normal distribution Φ and rearranged PROB. Updated CNST following *CODATA 2018*. Renamed the angular conversions. Changed the composing and cutting functionality of CC. Refined exiting *short integer* input. Expanded *App. D*. Specified maximum size of *long integers*. Changed keyboard adding $\sqrt[x]{}$, moving CPX, FIN, RBR, R↑, and SHOW, removing %. Renamed VANGLE to $V_{\sqrt[x]{}}$. Modified CPX, MATX, TRI, and X.FN. Rearranged *OMS* 1. Added some internal *DTs* to *App. B*; reduced the range of *long integer* results and *DP real* inputs to $10^{\pm 999}$. Defined the domains of e^X-1, IDIVR, LN(1+x), MOD, and RMD according to the *HP-42S*; modified PLOT and Σ+. Refined the *Addressing Tables*. Added a *DT* matrix for IDIVR. Refined the *Special Results* in *App. B*.

DRAFT

Version	Date	Release notes
0.13	30.11.19	Expanded the alpha keyboard and *App. I*. Modified CPX, INTS, MODE, PROB, STK, TEST, α●, SHOW, and STATUS. Refined the sorting order of *items*, ALL, CX→RE, MEM?, RE→CX, RBR, RM, SLVQ, and U→. Started filling *App. F* and *G*. Refined *App. 2*. Added a *long integer* example, CPXR?, LZ?, Δv_{Cs}, conversions of *hectares*, and a proposal for system status information.
0.14	7.3.20	Introduced *system flags* for status information. Split I/O. Added CATALOG'SYS.FL, PRINT, PROG, RANI#, VAR, auxiliary constants, some predefined *variables*, and an index in *App. I*. Changed keyboard swapping MODE and FLAGS, U→ and ⍋→, moving CPX, FILL, RBR, R↑, USER, α.FN, αINTL, √x̄, and 🖶, displaying PRINT, RMD, STATUS, x^2, and ':', and removing c/d, 🖶x, →SP, and →DP. Renamed DISP to DSP and SUMS to Σ, changed ↩ to ↶. Refined the addressing tables and *catalog* access, a b/c, ADV, BATT?, BITS, CATALOG'CHARS and 'MENUS, CLALL, CLFALL, CPX, EXP, GAP, INTS, I/O, MODE, NEIGHB, PARTS, PRIME?, P.FN, SHOW, STAT, STK, X.FN, αINTL, and α●. Deleted all 16-digit (i.e. *SP*) *DTs* as well as A…Z and the commands CLK12, CLK24, CPXi, CPXj, CPXRES, CPXR?, DBL?, DENANY, DENFAC, DENFIX, ENGOVR, FAST, IMPFRC, LZOFF, LZON, LZ?, MULT×, MULT·, POLAR, PROFRC, QUIET, RDX., RDX,, REALRE, RECT, SCIOVR, SLOW, SSIZE4, SSIZE8, →DP, and →SP. Corrected.
0.15	14.6.20	Added BESTF?, RANGE, RANGE?, REGIST, SNAP, and s(a), as well as errors 28 and 31 – 35. Changed DSZ and ISZ to comply with *HP-16C*. Changed keyboard shifting N, O, P, and Q, swapping ? and Z, moving CNST, CPX, FLAGS, RBR, RTN, R↑, VIEW, and 🖶, removing :, and adding MOD, ✓, and SNAP. Renamed DSP to DISP, CNST to CONST, CONST to CNST, ASL.BLK to ASLIFT, SSIZE to SSIZE8, TDM to TDM24, and the left and right sided probabilities. Refined ASSIGN, CATALOG, CNST, DISP, INFO, NEXTP, PRIME?, PROB, RBR, RESET, SHOW, SINC, STAT, U→, VIEW, x=+0?, x=−0?, y^x, α→x, ⍋, as well as *OMS* 6, *App. B*, *C*, and *E*, and some looping and statistical explanations. Reduced the maximum number of *local registers* from 100 to 99. Changed ALLSCI to ALLENG and RECTN to POLAR. Added *DT* matrices for powers. Corrected.

DRAFT

Version	Date	Release notes
0.16	3.11.20	Added torque and mmHg conversions, ISM, LOADV, x_{max} and x_{min}. Added UNDO to the *IOI*. Refined I/O and the descriptions of LOAD, LOADSS, RESET, and UNDO. Marked the not-undoable *items* in the *IOI*. Renamed the constants according to the *OM* and kicked them out of the *IOI.* Added USB. Refined *Basic Kinds of Program Steps*, *App. E* and *F.* Changed bit numbering from 1…64 to 0…63. Renamed SMODE? to ISM?. Refined IM, RE, SINC, TRI, WSIZE, X.FN and the labels of torque conversions. Added SINCπ and more vintage pictures. Expanded *App. G*. Refined the power matrix. Corrected.
0.17	16.3.21	Empty menus show up. Refined $\sqrt[3]{x}$, AUTOFF, BATT?, BITS, CLFALL, CLP, CLREGS, DSZ, e^x, FP, GTO., IP, ISZ, J/G, LOAD, LOCR, LOCR?, LN, MEM?, M.EDI, M.EDIN, NaN?, RCLEL, RDP, RSD, SAVE, SDL, SDR, TDISP, WSIZE, x=+0?, x=−0?, $\sqrt[x]{y}$, y^x, Σ+, Σ−, ^MOD, $\sqrt{x}$, 🖶r, labels in U→, the distributions, the *DT* matrices, *Sect. 3* and *4* and *App. 1*, as well as *App. B, E, F,* and *I*. Renamed ST.X etc. to X etc. Increased the number of local flags to 32. Added Chinese units of length, area, and mass to U→. Modified many conversions. Deleted M.OLD, added DELITM, J/G?, and 🖶x. Moved 'no' from 7 to /. Corrected.
0.18	18.4.21	Refined *Times*, the commands EXPON…, M.GET, M.PUT, M.R↕R, NBIN…, SAVE, STAT, and TDISP, *Sect. 3* of the *OM*, *App. B, E, F, H, and I*. Removed error 27, CATALOG'DIGITS, 'REGIST, and 'SYS.FL. Added ASSESS, CENTRL, NXTFIT, PLOT, and s_{mi}. Brought *wrap* and *grow* in the *status bar*. Corrected.
0.19	2.5.21	First release. – Introduced STATS. Refined *App. E* and *F*. Corrected.
0.20	25.9.21	Changed keyboard (re-) introducing DSP, x↕, and | |, making EXP and $\sqrt{x}$ primary, moving BITS, CPX, π, CONST, FILL, STK, TIMER, CLK, and SHOW, swapping I/O and LOOP, PRINT and TEST, SHOW, PARTS, and INTS, removing y^x and x^2 into EXP, and deleting RMD from the keyboard. Removed error 30; added errors 38, 39, and 40, the elliptic functions, HIDE, HIDE?, M.QR, Mπ→R, R→Mπ, and $Y_y(x)$. Refined ASSESS, BITS, CLK, COV, EIGVAL, EIGVEC, INDEX, LOADSS, LOADΣ, MATX, M.DIM, M.LU, M.NEW, RANGE, SAVE, SETDAT, SETTIM, s_{XY}, s(a), TONE, VIEW, X.FN, Σ+, Σ−, ∡→, two constants, the chapters about matrices, integration, some *DT* matrices, *App. 1, B, H,* and *I*. Corrected.

DRAFT

Version	Date	Release notes
0.21	14.1.22	Replaced the keyboard label TIMER by ⏱; in *AIM*, changed the label for αMATH, brought √‾ up (instead of √), e^, (,), and || for equations. Swapped CONST and π as well as STAT and Σ. Added a chapter to *Section 2*, an example for differential equations, ATAN2, CALC, CLBKUP, DRAW, ENDPMT, REDRAW, SQRT, ZOOM, and errors 47 - 50. Refined alphanumeric input, *local labels*, ALPHA, CLCVAR, FIN, INPUT, KEYG, KEYX, M.EDIN, PARTS, RCLEL, ROUNDI, SOLVE, TRI, TVM, VIEW, α●, systems of simultaneous equations, three constants, *OMS 4*, *App. 3, A, B, H,* and *I*. Removed SIGN from CPX. Renamed CEIL, CONJ, CROSS, DOT, FLOOR, LN, LNβ, LNΓ, LN(1+x), $LOG_x y$, and SIGN to lower case Latin, LOG_{10} to lg, LOG_2 to lb, DENMAX to D.MAX, and PRINT to PRINTS. Defined *names* more precisely. Corrected.
0.22	13.9.22	Moved CPX, FLAGS, LBL, MODE, STK, U→, and α.FN on the bezel, put SF and CF on the free space. Renamed BITS to BIT, FLAGS to FLAG, PARTS to PART, and relabeled PRINT to 🖶. Labeled the menu keys. Removed the labels of the prefixes. Added FRCSRN, READP, WRITEP, histogram commands, an appendix about graphics, a chapter about I/O, examples for solid angle, elastostatics, and tool life. Renamed GTO.▲ to GTO▲ and GTO.▼ to GTO▼, TIMER RESET to T.RES. Removed RECV, SEND, XIN, XOUT, and the *program top* indicator. Refined 0.1s, ADD, AGRAPH, ASSIGN, AUTXEQ, ATAN2, CENTRL, CLLCD, LOADP, SOLVE, STATUS, TIMER, VARMNU, equation editing, 3 letters, 4 conversions, fraction display, *names*, *AIM*, functionalities of ⬅, EXIT, and R/S, *App. 1, 5, E, F, G,* and *H*. Corrected.
0.23	3.12.22	Relabeled *WP 43S* to *WP43*. / is replaced by ÷ on the keyboard. Added omicron in *AIM*. **Pilot run performed and distributed in 2022-10.** 28 pilot serial numbers are known (cf. App. K). Two more were distributed, so we know of 30. *Michael* tells 35 were produced and delivered (the maximum serial number we know is 00044 – this looks like a scrap/rework rate of 20% at least for that run featuring standard *DM42* electronic hardware in the summer of 22). Introduced $\sqrt{1+x^2}$, status indicators 4 and 8, and *virtual keyboards* for RBR, FBR, STATUS, and TIMER. Elaborated about histograms. Renamed TIMER ADD to T.Σ+. Refined *App. 4*, AGRAPH, CPX, erf, erfc, EXITALL, EXPT, J→D, LNΓ, MANT, MATR, PART, V⁄, x!, αMATH, Γ(x), ζ(x), ⁄, and ⬅. Corrected.
0.23.3	21.1.23	Implemented DELITM, SAVE and LOAD of *variables*, confirmation for CLP, RBR for *variables*, and A…D and I…L keys in RBR. Reintroduced (a new) error 27. Fixed simulator crash when restarted after closed while a browser was open.

DRAFT

Version	Date	Release notes
0.23.4	6.2.23	Implemented FASTFN and CLP *menu*. Corrected DELITM, PIXEL, AGRAPH, string concatenation, Σ+ memory leakage, CNST indirection, and possible crash of GTO/XEQ after LOAD. Optimized some common functions.
0.23.5	23.2.23	Implemented VARMNU, NUM.IN and ALP.IN. Fixed possible crash of CLP, W_p, and matrix division; corrected TAN in MULπ, KEYG and KEYX (removed K as *alpha register*, cf. 0.7), DECOMP, M.LU; corrected *complex* matrix subtraction, eigenvalues and eigenvectors, y^x, and β; cube root with FASTFN set, COMB and PERM, special cases of lnΓ, g_d, →R, and DECOMP.
0.23.6	23.2.23	As 0.23.5 but with updated manuals.
0.23.7	11.3.23	Added ACTUSB (also on the calculator keyboard), DMCP, and DUMP. Refined NEXTP; also VARMNU, x→α, and handling of empty strings to meet the specs of *HP-42S*. Elaborated on I/O. Corrected.
0.23.8	11.3.23	Corrected a fatal bug in the simulator. Else unchanged.
0.23.9	12.3.23	Added the angular unit *mil*.
0.23.10	25.3.23	Implemented →BIN, →DEC, →OCT, and →HEX in INTS. Refined $(-1)^x$, VARMNU, and the *QRG*. Revised and corrected.
0.23.11	8.4.23	Refined BIT, CLPALL, INTS, PAUSE, RCL, VARMNU, VIEW, #, the comparisons, *App. 4*, and the *QRG*. Corrected.
0.23.12	5.5.23	Added LOADST and SAVEST. Refined handling of undefined *variables*, I/O, SOLVE, VARMNU, x→α, ∫, the *status bar*, *OMS 1, 3,* and *4*, *ReMS 3*, *App. B* and *J*. Corrected.
0.23.13	1.6.23	Added error 53. Removed error 19, FLASH, RAM, PRCL, and PSTO. Reorganized access to *FM*. Implemented READP and WRITEP. Refined CLREGS, LOADP, LOADR, LOADSS, LOADV, LOADΣ, PROGS, SNAP, SOLVE, *App. I*, creating *user menus*, and displaying global labels. Corrected.
0.24	1.7.23	Added DISK? and *character* translation for editing programs on a computer. Refined head text of *IOI*, EXITALL, INFO, INT?, J/G?, SNAP, x!, α●, some names of conversions, and *stack* diagrams. Removed FLASH? and ⤒. Corrected.
0.24.1 and .2	8.9.23	Expanded the *character* translation table. Removed 🖷LCD. Added BUZZ, CONFIG, MyPFN, PLAY, VOL, VOL+, VOL-, VOL?, and V.ALL. Refined BACK, CASE, EXITALL, INFO, I/O, MVAR, NOP, P.FN, P.FN2, SKIP, TICKS, and VARMNU. Filled *App. H.* Changed the virtual keyboard for *AIM*. Corrected.

DRAFT

Version	Date	Release notes
0.24.3	2.12.23	Specified *startup default* contents for Myα and MyPFN. Added CLKEYS, CPXSLV, MEDDEV, MEDIAN, PERCTIL, Q_3-Q_1, REM, XEQ+1, x_{ma}–x_{mi}, and appendices for *Sound* and history. Refined ASSIGN, CLR, STAT, U→, $\bar{x}_H$, x_{max}, x_{min}, $\bar{x}_{RMS}$, ∫, *user mode*, the *IOI, App. A, F,* and *J*. Renamed LOAD… to Load… and HIST to HISTOG. Deleted PRINTS and SLOW. Removed printer hardware references. Corrected.
0.25	23.12.23	Added CLMALL, CLMyM, CLMyP, CLMyα, CLMy…, CLVALL, day→s, DELMALL, DELVALL, s→day, and TIME:. Refined *dates* and the backplate print, CLALL, CLR, CLMENU, CPXSLV, DELITM, RESET, and U→. Renamed XEQ+1 to XEQ.SKP. Corrected.
0.25.1	25.1.24	Added SLVC. Refined lettered *flags*, DELITM, DELMALL, DELVALL, STATUS, U→, x→α, and *App. G*. Corrected.
0.25.2	30.1.24	Firmware corrected.
0.26	7.7.24	Added MEDIAN and related commands. Split STAT in two *menus*. Created FIT. Refined and corrected. Modified 1/x.
0.26.1	28.7.24	Expanded yes/no decisions. Refined median commands. Added :, CH, and QU for *AIM*. Rearranged and corrected.
0.26.2	21.8.24	Added AVIEW and PROMPT, refined P.FN, P.FN2, VIEW, error 11, and *App. E*. Limited calculator input for NEXTP and PRIME?. Updated bezel and backside. Refined and corrected.
0.26.3	22.9.24	Added M.FIND, M.CMAX, and M.CMIN. Refined *App. E, H, I, and J*, AGRAPH, CONVG?, ERR, INDEX, INPUT, MSG, M.DIM, M.EDIN, NEXTP, RANGE, RDP, RESET, SIM_EQ, SOLVE, %ile, ∫, and the *QRG*; renamed IΓ to Γ; rearranged *ReM* chapters and corrected.
0.27	9.11.24	Expanded the virtual keyboard for labels. Made 8 *stack* levels default. Swapped some *OM* chapters. Refined *App. 1, B, E, F,* and *I*. Added e and i. Refined and corrected.
0.27.1	16.12.24	Added FACTOR, many kitchen volumes, two examples to *OMS 5*, one to *App. 4*, and a chapter to *OMS 2* and *App. J*. Refined U→, X.FN, and *App. A*. Rephrased p. 1. Refined and corrected.
0.27.2	19.12.24	Corrected.
0.28	1.5.25	Updated CONST following *CODATA 2022*. Renamed RCL↑ to RCLMAX, RCL↓ to RCLMIN, RCLS to RCLSTK, and the respective store commands. Removed page background color. Refined and corrected.

DRAFT

Version	Date	Release notes
0.28.1	9.5.25	Refined the *IOI* to contain (almost) all *items*. Added another example. Refined and corrected.
0.28.2	15.7.25	Split LoadPE into LoadPr and LoadEq. Extended LoadR and the *OM* and *ReM* indices. Changed RTN+1 to RTN.SKP. Deleted LOADST, refined LOAD… and I/O. Put CONST in CATALOG. Added Indian output format, generic EDIT, auxiliary constants, keystroke numbers, *App. 4* about PLOT, two chapters to *App. H*, two to *App. I*, one to *App. J*, *App. K* about the pilots, and *App. N*. Improved the explanations of CPX?, OFF, SAVEST, and *App. C*. Refined and corrected.
0.28.3	22.8.25	Added EDIT and ! to the bezel, moved USB and DMCP. Elaborated about angles. Refined and corrected.

DRAFT

APPENDIX N: INDEX OF NAMES, CALCULATORS, COMPANIES, ETC.

MS Word generated the following index (almost) automatically, so the sorting order below neither complies to the *IOI* above nor follows a comprehensible logic. I changed it only where its sequence was massively counterintuitive. This index may be helpful nevertheless, since

1. showing all *items* provided, though in another order some may be more accustomed to (so you know which *names* you shall not use when creating new *user items*), and
2. covering the entire text above plus the *Background Facts and Considerations* below.

Keyboard labels are printed in blue or gold according to their appearance there, *submenus* (as well as EXP and TRI) in black. Everything not being an *item* provided is printed in dark blue. For living *persons*, only their prenames are listed, while historic persons or fictional names are listed fully.

Please check also the *Table of Contents*

.

- 106
#B 112
#DEC 112

% 109
%+MG 109
%ile 109
%MRR 109
%T 109
%Σ 109

(-1)^x 105
(χ²)^-1 105

/ 106
[M]^-1 105
[M]^T 105
^MOD 106
| | 109
|M| 108
|x| 108
+ 105
+/- 105
±∞? 105
× 106
×MOD 106

√x 109
∛x 20
ˣ√y 96
∢ 110
∢→ 111, 150
∫ 110
∫f 110, 142
∫fdx 110, 141

↑Lim 108
→ 106
→BIN 106
→D.MS 106

DRAFT

→DATE 106
→DEC 106
→DEG 106
→GRAD 106
→H.MS 107
→HEX 106
→HR 106
→INT 107
→MIL 107
→MULπ 107
→OCT 107
→POL 107
→RAD 107
→REAL 107
→REC 108
↓Lim 108
↯ 108

°C→°F 20
°F→°C 20

145
🖷# 112
🖷ADV 111
🖷CHAR 111
🖷DLAY 111
🖷MODE 111
🖷PROG 111
🖷r 111, 112
🖷REGS 111
🖷STK 112
🖷TAB 112
🖷USER 112
🖷WIDTH 112
🖷Σ 112

1/x 20
10^x 20
1COMPL 20
2^x 20
2COMPL 20
∞ 109, 119, 199
-∞ 106, 119, 199

a 20, 114
A 20
A: 24, 146
$a_{\oplus}$ 24, 114
a_0 20, 114
ABS 20
ac→m^2 21
ac_{US}→m^2 21
ACC 20
ACOS 20
ActUSB 20
ADD 21, 104
ADM 21
ADV 21, 141
AGM 21
AGRAPH 21
ALL 21
ALLENG 21f, 130
ALP.IN 22, 130
ALPHA 22, 129
a_{Moon} 22, 114
AND 22
ANGLE 22
ANGLES 22
arccos 22
arcosh 22
arcsin 22
arctan 23, 24
artanh 23
ASIN 23
ASLIFT 23, 131
ASR 23
ASSESS 143
ASSIGN (ASN) 23
ATAN 24
atm→Pa 24
au→m 24
AUTOFF 24, 131
AUTXEQ 130
AVIEW 24

B 24
Babylon 129
BACK 24
bar→Pa 24
BATT? 25
bbl→m^3 25
BC? 25
BEEP 25
BeginP 25
BestF 25
BestF? 26

DRAFT

Binom^{-1} 26
Binom$_{\blacktriangleleft}$ 26
Binom$_{\blacktriangle}$ 26
Binom: 26, 145
Binom$_{p}$ 26
BIT 26, 141
B_n^* 26
B_n 26
Braun B-4
BS? 26
Btu→J 26
BUZZ 26

c 27, 114
C 26
c_1 27, 114
c_2 27, 114
C47 6
cal→J 27
CALC 27
Carl F. Gauss 113
CARRY 27, 129
CASE 27
CATALOG 28
Cauch^{-1} 28
Cauch$_{\blacktriangleleft}$ 28
Cauch$_{\blacktriangle}$ 28
Cauch: 28, 145
CauchF 28
Cauch$_{p}$ 28
CB 28
CEIL 28
CENTRL 28
César R. 329
CF 29
CHARS 29
chĭ→m 29
CLall 29
Classics 3
CLBKUP 29
CLCVAR 29
CLFall 29
CLK 29, 142
CLKEYS 29
CLLCD 29
CLMall 29
CLMENU 29
CLMy... 29, 142
CLMyM 30
CLMyP 30
CLMyα 30
CLP 30
CLPall 30
CLR 30, 142
CLREGS 30
CLSTK 30
CLVall 30
CLX 30
CLΣ 30
cn(υ,m) 30, 319
CNST 30
CODATA 2022 113
COMB 31
Commodore 325
CONFIG 31, 135f
CONJ 31
CONST 31, 142
CONVG 31
CORR 32, 72
cos 32
cosh 32
COV 32
cp_I→mℓ 32
cp_{UK}→mℓ 32
cp_{US}→mℓ 32
CPX 32, 142
CPX? 32
CPXRES 32, 128
CPXS 32
CPXSLV 32
cross 33
ct→g 33
ct→kg 33
cùn→m 33
curve fitting 335
cwt→kg 33
CX→RE 33

D 33
D.MAX 37
D.MS 37
D.MS→ 37
D.MS→D 37
D.MY 37
D→D.MS 37
D→J 38

DRAFT

D→MIL 38
D→R 38
data types 189
DATE 33
DATE→ 33
DATES 33
David J. 6
DAY 34
day→s 34
dB→fr 34
dB→pr 34
DBL/ 34
DBL× 34
DBLR 34
DEC 34
DECIM. 34, 130
DECOMP 34
DEG 34
DEG→ 35
DelITM 35
DelMall 35
DelVall 35
DENANY 35, 128
DENFIX 35, 128
DENMAX 35
DET 35
Dieter Rams B-4
DISK? 35
DISP 35, 142
DM42 1, 3, 180, 183f, 186, 207, 244ff, 248, 251, 339, B-6, -31
DMCP 35, 192, 246-249, 254, B-4
DMCP 35, 248, 254
DMY 35, 128
dn(u,m) 35, 319
dot 36
DRAW 36
DROP 36
DROPy 36
DSE 36
DSL 36
DSP 36
DSTACK 37
DSZ 37
DUMP 37

e 38, 114
e 40, 114
E(m) 40, 319
E(φ,m) 40
E: 40, 146
e^x 40
e^x-1 40
EDIT 38
EIGVAL 38
EIGVEC 38
Eisberg 302
Ellipt 38, 148, B-3
END 38
ENDP 38
ENDPMT 38
ENG 38
ENORM 39
ENTER↑ → ENTER↑
ENTRY? 39
EQ.DEL 39
EQ.EDI 39, 142
EQ.NEW 39
EQN 39, 142
Eratosthenes 327
erf 39
erfc 39
Eric R. 6
Eric S. 342
ERR 39
ET66 B-4
EVEN? 39
EXIT → EXIT
EXITall 40
EXP 40, 142
ExpF 40
$Expon^{-1}$ 40
$Expon_{\blacktriangle}$ 40
$Expon_{\blacktriangle}$ 40
Expon: 40, 145
$Expon_p$ 40
EXPT 40

F 40, 114
f' 43, 142
f'(x) 43
f'' 43, 142

DRAFT

f''(x) 43
F&p: 43, 146
$F^{-1}(p)$ 42
$F_{\Delta}(x)$ 42
$F_{\perp}(x)$ 42
F: 43, 145
FACTOR 41, 327
FASTFN 41, 130
FB 41
FBR 41
FC? 41
FC?C 41
FC?F 41
FC?S 41
FCNS 41
FF 41
FIB 41
FILL 41
FIN 41, 143
FIT 41, 143
FIX 41
FLAG 143
FLAGS 41
FLOOR 42
fm.→m 42
FP 42
$F_p(x)$ 42
FP? 42
fr→dB 42
FRACT 42, 128
FRCSRN 42, 128
Free42 248, 265,
B-8
FS? 42
FS?C 42
FS?F 42
FS?S 42
ft.→m 42
ft_{US}→m 42
FV 42
fz_{UK}→m^3 42
fz_{US}→m^3 42
fz_{US}→mℓ 42
F_{α} 43, 114
F_{δ} 43, 114

G 43, 115
$g_{\oplus}$ 45, 115
g→ct 45
g→oz 45
g→trz 45
G_0 43, 115
GAP 43
GaussF 44
G_C 44, 115
GCD 44
g_d 44
g_d^{-1} 44
g_e 44, 115
$Geom^{-1}$ 44
$Geom_{\Delta}$ 44
$Geom_{\perp}$ 44
Geom: 44, 145
$Geom_P$ 44
gl_{UK}→m^3 44
gl_{US}→m^3 44
$GM_{\oplus}$ 44, 115
GRAD 44
GRAD→ 44
GRAMOD 45
GROW 45, 129
GTO 45
GTO. 45
GTO↑ 45
GTO↓ 45

h 45, 115
ℏ 47, 115, B-10, -23
ha→ac_{US} 46
ha→ac 46
ha→m^2 46
Heathkit 260
HIDE 46
HIDE? 46
HISTO 46
HISTOG 46, 146
HISTOX 46
HISTOY 46
H_n 46
H_{nP} 46
HP 15f, 202, 259, 261, 267, 280, 329, B-4f, -19
HP-12C 267, 335
HP-15C 16, 259, 329, B-22
HP-16C 15, 23,

DRAFT

37, 49, 54, 58, 67, 76ff, 84, 92, 95, 112, 129, 192, 215, 221, 223, 240, B-32
HP-19BII 267
HP-19C 16, 37, 49
HP-20b B-4
HP-21S 42, 91, 105, 215, 223f, 240
HP-28C/S 22, 261
HP-29C 16, 37, 49
HP-30b B-4
HP-34C 16, 209, 240, 329
HP-35 259, 263, 5
HP-35S 35
HP-41C 16, 188, B-22
HP-41CX B-5
HP-42S 15f, 21f, 25, 45, 47, 51, 62, 77f, 88, 97, 99, 129ff, 215, 221, 226, 240, 328f, B-6, -19, -22, -27, -29
HP-48 188, B-29
HP-50g 261, B-22
HP-55 16
HP-67 16, 37, 49
HP-71 66
HP-97 16
hp_E→W 46
HPLOT 146
hp_M→W 46
hp_{UK}→W 46
$Hyper^{-1}$ 46
$Hyper_{\triangle}$ 46
$Hyper_{\triangle}$ 46
Hyper: 47, 145
$Hyper_p$ 46
HypF 47

I 47
I- 50
i 14, 50, 115, 337, 11, 13, 24
i%/a 50
I/O 50, 143
I+ 50
IDIV 47
IDIVR 47
IGN1ER 47, 131
iHg→Pa 47
Im 47
in.→m 48
in.→mm 48
INC 47
INDEX 47
INFO 47, 143
INPUT 48
INT? 48
Integrator 329
INTING 48, 131
INTS 48, 143
INVRT 48
IP 48
ISE 49
ISG 49
ISM 49
ISM? 49
ISZ 49
I_{xyz} 50
$I\Gamma_p$ 101
$I\Gamma_q$ 101

J 50
J- 50
J/G 50
J/G? 50
J+ 50
J→Btu 50
J→cal 50
J→D 51
J→Wh 51
Jake S. 342
jīn→kg 50
Jobs, Steve B-4
$J_y(x)$ 50

k 51, 115
K 51
K(m) 52, 319
KEY 51
KEY? 51

DRAFT

KEYG 51
KEYX 51
kg→cwt 52
kg→jīn 52
kg→lb. 52
kg→mi 52
kg→oz. 52
kg→s.t 52
kg→scw 52
kg→sto 52
kg→ton 52
kg→trz 52
K_J 52, 115
km→nmi 52
KTYP? 52

L 53
L.INTS 56
L.R. 57
ℓ→gal_{US} 57
ℓ→gal_{UK} 57
ℓ→qt. 57
LASTx 53
lb.→kg 53
lbf→N 53
LBL 53
LBL? 53
LCM 53
LEAD.0 129
LEAP? 53
lft→Nm 53
lg 20, 53
$LgNrm^{-1}$ 54
$LgNrm_{\blacktriangle}$ 54
$LgNrm_{\blacktriangle}$ 54
LgNrm: 54, 145
$LgNrm_p$ 54
lĭ→m 54
liă→kg 54
LinF 54
LJ 54
L_m 54
$L_{m\alpha}$ 54
ln 54
ln(1+x) 55
lnβ 54
lnΓ 55
LoadAll 55, 241, 249
LoadEq 55
LoadPr 55
LoadR 55
LoadSS 55
LoadV 56
LoadΣ 56
LocR 56
LocR? 56
LogF 56
$Logis^{-1}$ 56
$Logis_{\blacktriangle}$ 56
$Logis_{\blacktriangle}$ 56
Logis: 56, 145
$Logis_p$ 56
$log_x y$ 56
LOOP 56, 144
LOWBAT 56, 130
l_{PL} 56, 115
ly→m 57

M.↑ 65
M.→ 65
M.↓ 65
M.← 65
M.CMAX 61
M.CMIN 61
M.DELR 62
M.DIM 62
M.DIM? 62
M.DY 62
M.EDI 62
M.EDIN 62
M.EDIT 62, 144
M.FIND 62
M.GET 63, 64
M.GOTO 63
M.GROW 63
M.INSR 63
M.LU 63, *9*
M.NEW 64
M.PUT 63, 64
M.QR 64, *9*
M.R⇄R 64, 226
M.SIMQ 58, 64, 140, 144
M.SQR? 64
M.WRAP 65
m: 65, 147

DRAFT

$M_{\oplus}$ 65, 116
$M_{\odot}$ 65, 116
m→au 65
m→chǐ 65
m→cùn 65
m→fēn 65
$m→ft_{US}$ 65
m→ft. 65
m→ftm. 65
m→in. 65
m→l.y. 65
m→lǐ 65
m→mi. 65
m→nmi 65
m→pc 65
m→pt. 65
m→yd. 65
m→yǐn 65
m→zhàng 65
$m^2→ha$ 57
$m^3→bbl$ 57
MANT 58
Marcus v. C. 342
MASKL 58
MASKR 58
Mat.A 58, 136
Mat.B 58, 136
Mat.X 58, 136
Mat_A 58, 135
Mat_B 58, 135
Mat_X 58, 135
Mathworld 318, 323, 325f
MATR? 58
MATRS 58
MATX 58, 144
max 59
MDY 59, 128
m_e 59, 115
MEDIAN 59
MEM? 58, 59
MENU 59
MENUS 59
mi.→km 59
mi.→m 59
Michael S. 342, 347, B-6
Michalski 329
MIL 59
MIL→ 59
MIL→D 59
MIL→R 59
min 59
MIRROR 59
$mℓ→cp_I$ 59
$mℓ→cp_{UK}$ 59
$mℓ→cp_{US}$ 59
$mℓ→pt_{UK}$ 60
$mℓ→fz_{UK}$ 60
$mℓ→fz_{US}$ 60
$mℓ→pt_{UK}$ 60
$mℓ→pt_{US}$ 60
$mℓ→tb_I$ 60
$mℓ→tb_{UK}$ 60
$mℓ→Tb_{US}$ 60
$mℓ→ts_I$ 60
$mℓ→ts_{UK}$ 60
$mℓ→ts_{US}$ 60
mm→in. 60
mm→pt. 60
mmHg→Pa 60
M_{Moon} 115
m_n 60, 116
m_n/m_p 60, 116
MOD 60
MODE 60
MoHPC 255, 262, B-4
MONTH 60
Mosig 329
m_p 60, 116
m_p/m_e 60, 116
m_{PL} 60, 116
MSG 61
m_u 61, 116
$mŭ→m^2$ 61
m_uc^2 61, 116
MULT× 61, 130
MULπ 61
MULπ→ 61
MVAR 61
MyMENU 61, 144
MyPFN 61, 144
Myα 61, 144
$m_μ$ 61, 116
Mπ→R 61

DRAFT

N→lbf 68
N_A 65, 116
NaN 66, 116, 130, 196, 199, 213, B-15
NaN? 66
NAND 66
$NBin^{-1}$ 66
$NBin_{\blacktriangle}$ 66
$NBin_{\blacktriangle}$ 66
NBin: 66, 145
$NBin_p$ 66
nBINS 66
NEIGHB 66
NEXTP 67
Nils Holgersson 149
NIST 113, 161
Nm→lbft 67
nmi.→km 67
nmi.→m 67
NOP 67
NOR 67
$Norml^{-1}$ 67
$Norml_{\blacktriangle}$ 67
$Norml_{\blacktriangle}$ 67
Norml: 67, 145
$Norml_p$ 67
NOT 67
NPER 67
NUM.IN 67, 130
Nuts 3
NXTFIT 23, 67
nΣ 68

ODD? 68
OFF 68
OR 68
OrthoF 68
Orthog 68, 148
OVERFL 68, 129
oz.→g 68
oz.→kg 68

P.FN 72, 145
P.FN2 72, 145
P: 72, 147
P_0 68, 116
Pa→atm 68
Pa→bar 68
Pa→iHg 68
Pa→mmH 69
Pa→psi 69
Pa→torr 69
ParabF 68
PART 68, 144
Paul D. 342, B-6
PAUSE 68
pc→m 69
PER/a 69
PERM 69
PGMINT 69
PGMSLV 69
Pioneers 3
PIXEL 69
PLAY 69, 174
PLOT 69, 143
PMT 70
P_n 70
POINT 70
$Poiss^{-1}$ 70
$Poiss_{\blacktriangle}$ 70
$Poiss_{\blacktriangle}$ 70
Poiss: 70, 145
$Poiss_p$ 70
POLAR 70, 128
PopLR 70
PowerF 70
pr→dB 71
prime sieve 327
PRIME? 71
PRINT 145
PROB 71, 145
PROG 71
PROGS 71
PROMPT 24, 71, 174
PROPFR 71, 128
psi→Pa 71
pt.→m 71
pt.→mm 71
PTB 113
pt_{US}→mℓ 71
pt_{UK}→mℓ 71
PUTK 71
PV 71

Q_3-Q_1 72

DRAFT

qt.→ℓ 72
QUIET 72, 131

R_∞ 80, 117
R 72, 117
$R_\oplus$ 80, 117
$R_\odot$ 80, 117
R↑ 80
R→D 80
R→MIL 80
R→Mπ 80
R↓ 80
R47 6
RAD 72
RAD→ 72
Rams, Dieter B-4
RAN# 73
random numbers 334
RANGE 72
RANGE? 72
RANI# 73
RBR 73
RCL 73
RCL/ 74
RCL+ 74
RCL× 74
RCLCFG 73
RCLEL 73
RCLIJ 74
RCLMAX 74
RCLMIN 74
R-CLR 79
RCLSTK 74, 88, 212, 327, B-28
R-COPY 79
RDP 74
Re 74
r_e 74, 117
RE→CX 76
Re≷Im 76
READP 75
REAL? 75
REALDF 75
REALS 75
REDRAW 75
REM 75, 168, 304-307, 310, 333
RESET 75
Richard Nelson B-19
Richard Ottosen 342, B-6
RJ 76
R_K 76, 117
RL 76
RLC 76
RM 76
RM? 77
RMD 77
R_{Moon} 77, 117
RNORM 77
RootF 77
ROUND 78
ROUNDI 78
RPL 15, 261, 263, B-21f, -28
RPN 16, 18, 37, 172, 257f, 260f, 263, 267, 327, B-4, -19, -21f, -27f
RR 78
RRC 78
RSD 78
R-SORT 80
RSUM 78
R-SWAP 80
RTN 79
RTN.SKP 79
RUNIO 79, 129
RUNTIM 79, 129

s 80
s(a) 89
S.INTS 89
s.t→kg 83
s→day 89
s→year 89
Sa 81, 117
SAVE 29, 55, 81, 192f
SAVEST 81
Sb 81, 117
SB 81
SCI 81
scw→kg 81

DRAFT

SDIGS? 81
SDL 81
SDR 81
Se^2 82, 117
Se'^2 82, 117
SEED 82
SETCHN 82
SETDAT 82
SETEUR 82
SETIND 82
SETJPN 82
SETSIG 82
SETTIM 82
SETUK 82
SETUSA 82
Sf^{-1} 82, 117
SF 82
SHOW 83
sign 49, 83, 170, 192, 194
SIGNMT 83
SIM_EQ 83
sin 83
sinc 84
Sinclair 326
sinh 84
SKIP 84
SL 84
SLVC 84
SLVQ 85, *264*, 20
SM 13, 35, 186, 188, 250, 252, 254, 339f, B-6
s_m 85
s_{mi} 85
s_{mw} 85
sn(υ,m) 85, 235, 319
SNAP 86
SOLVE 86
Solver 87, 142, 328
SOLVING 87, 131
SPCRES 87, 130
SPEC? 87
Spices 3, 16, 16
SR 87
SSIZE? 87
SSIZE8 87, 130
STAT 87, 146
STATS 87
STATUS 87
Steve Jobs B-4
Stings 3
STK 87, 146
STO 87
STO— 88
STO/ 88
STO+ 88
STO× 88
sto→kg 88
STOCFG 87
STOEL 87
STOIJ 87
STOMAX 88
STOMIN 88
STOP 88
STOS 88
STRI? 88
STRING 88
SUM 88
s_w 88
s_{xy} 89
SYS.FL 89

T 89
$t^{-1}(p)$ 91
$t_{\Delta}(x)$ 91
$t_{\Delta}(x)$ 91
T.RES 75, 90, 92
t: 92, 145
t↗ 92
T_0 89, 117
tan 89
tanh 89
Taylor B-7
$tb_I \rightarrow m\ell$ 89
$tb_{UK} \rightarrow m\ell$ 89
$Tb_{US} \rightarrow m\ell$ 89
TDISP 90
TDM24 90, 128
TEST 90, 146
Thomas K. 333
TICKS 90
TIME 90
Time: 90, 147
TIMER (⏱) 90

DRAFT

TIMES 90
T_n 91
Tom Osborne B-5
ton→kg 91
TONE 91
TOP? 91
tor→Pa 91
$t_p(x)$ 91
t_{PL} 91, 118
T_{PL} 91, 117
TRACE 91, 129
TRANS 87, 91
TRI 91, 146
troz→g 91
trz→kg 91
ts_I→mℓ 91
ts_{UK}→mℓ 91
ts_{US}→mℓ 91
TVM 91, 143, 335

U→ 92, 146
ULP? 92
U_n 92
UNDO (↶) 92
UNITV 92
UNSIGN 92
USB 92, 130, 180, 251
USER 92, 112, 129

V.ALL 93
V: 93, 147
V↯ 94
Valentin A. 261, 331
VAR 93
VARMNU 93
VARS 93
VERS? 93
VIEW 93
V_m 93, 118
VMDISP 93, 131
VOL 93
VOL- 93
VOL- 93
VOL? 93
VOL+ 93
Voyagers 3, 16, 263, B-5

W^{-1} 94
Walter M. Kolb 280
W→hp_E 95
W→hp_M 95
W→hp_{UK} 95
WDAY 94
$Weibl^{-1}$ 94
$Weibl_{\blacktriangle}$ 94
$Weibl_{\vartriangle}$ 94
Weibl: 94, 145
$Weibl_p$ 94
Wh→J 94
WHO? 94
William Kahan 194, 329
W_m 94
WolframAlpha 199, 264
Woodstocks 3
word size 192
W_p 94
WP 31S 230, B-19, -27, -29
WP 34S 15f, 71, 95, 137, 190, 199, *207*, 215, 225, 230, 259, 265, 329, 342, B-4, -8, -14, -19, -27
WP 43S B-4f
WRITEP 94
WSIZE 95
WSIZE? 95

X 95
$\hat{x}$ 95
$\bar{x}$ 95
$\bar{x}_G$ 96
$\bar{x}_H$ 96
$\bar{x}_{RMS}$ 96
$\bar{x}_W$ 96
x < ? 98
x = ? 98
x ≠ ? 98
x = -0? 98
x = +0? 98
x > ? 98, 174, 222
x ≈ ? 98
x ≦ ? 98

DRAFT

x ≥ ? 98
x! 97
X.FN 96, 147
x: 97, 147
x→DATE 97
x→α 97
x≷ 97
x≷y 97
x^2 95
x^3 95
XEQ 95
XEQ.SKP 96
x_{max} 96
x_{min} 96
XNOR 96
XOR 96

Y 98
ŷ 98
Y.MD 99
y≷ 99
y^x 99
yd.→m 98
YEAR 98
year→s 98
yǐn→m 98
YMD 99, 128
$Y_y(x)$ 99

Z 99
z≷ 99
Z_0 99, 118
zhàng→m 99
ZOOM 23, 99

α 99, 118
α● 100, 150
A...Ω 100, 149
α.FN 100, 149
α→x 100
αCAP 99, 129
αINTL 99, 148
αLENG? 99
αMATH 99, 149
αPOS? 100
αRL 100
αRR 100
αSL 100
αSR 100

β(x,y) 100, 175, 228

γ 101, 118
Γ(x) 101
γ_{EM} 101, 118
γ_p 101, 118
γ_{xy} 101
Γ_{xy} 101

Δ% 101
δx 101
Δx 101
$\Delta\nu_{Cs}$ 101, 118

ε 101
ε_0 101, 118
ε_m 102
ε_p 102

ζ(x) 102
Z(φ,m) 102, 319

λ_C 102, 118
λ_{Cn} 102, 118
λ_{Cp} 102, 118

μ_0 102, 119
μ_B 102, 119
μ_e 102, 119
μ_e/μ_B 102, 119
μ_n 102, 119
μ_p 102, 119
μ_u 102, 119
μ_μ 102, 119

π 102
Π(n,m) 103, 319
Π_n 102

σ 103
Σ 103, 150
Σ- 104
Σ1/x 103
$\Sigma 1/x^2$ 103
Σ1/y 103
$\Sigma 1/y^2$ 103
Σ+ 104
σ_B 103, 119
$\Sigma \ln^2 x$ 103

DRAFT

Σln²y 103
Σlnx 103
Σlnxy 103
Σlny 103
Σlny/x 103
Σ_n 103
σ_w 103
Σx 88, 104, B-11
Σx/y 104
Σx² 104
Σx²/y 104
Σx²lny 104
Σx³ 104
Σx²y 104
Σx⁴ 104
Σxlny 104
Σxy 104
Σy 88, 104, B-11
Σy² 104
Σylnx 104

Φ 104, 119
Φ_0 104, 119

$\chi^2_{\blacktriangle}(x)$ 105
$\chi^2_{\blacktriangle}(x)$ 105
$\underline{\chi^2}$: 105, 145
$\chi^2_p(x)$ 105

ψ(υ,m) 105, 319

ω 105, 119, 179, B-11, -13, -25

≈ 99, 149
• 100, 150
← 15, 30, 164, 168, 211, 239, B-1
A 99, 148
CC 15, 33, 76, 167, B-1, -20
ENTER↑ 15, 165f, 168-172, 210, 241, 244f, 247, 249, 259ff, 267, B-1, -4f, -19, -28f, -39
EXIT 15, 23, 39, 51, 67, 86, 110, 134, 137, 139, 166, 172, 213, 244f, 247, 249, 254, 270, 328, B-1, -29

(The index may be slightly off – it will be updated before v1.0)

- 106
/ 106
+ 105
+/- 105
× 106
√x 109
1/x 20
e^x 40
ENTER↑ → ENTER↑
EXIT → EXIT

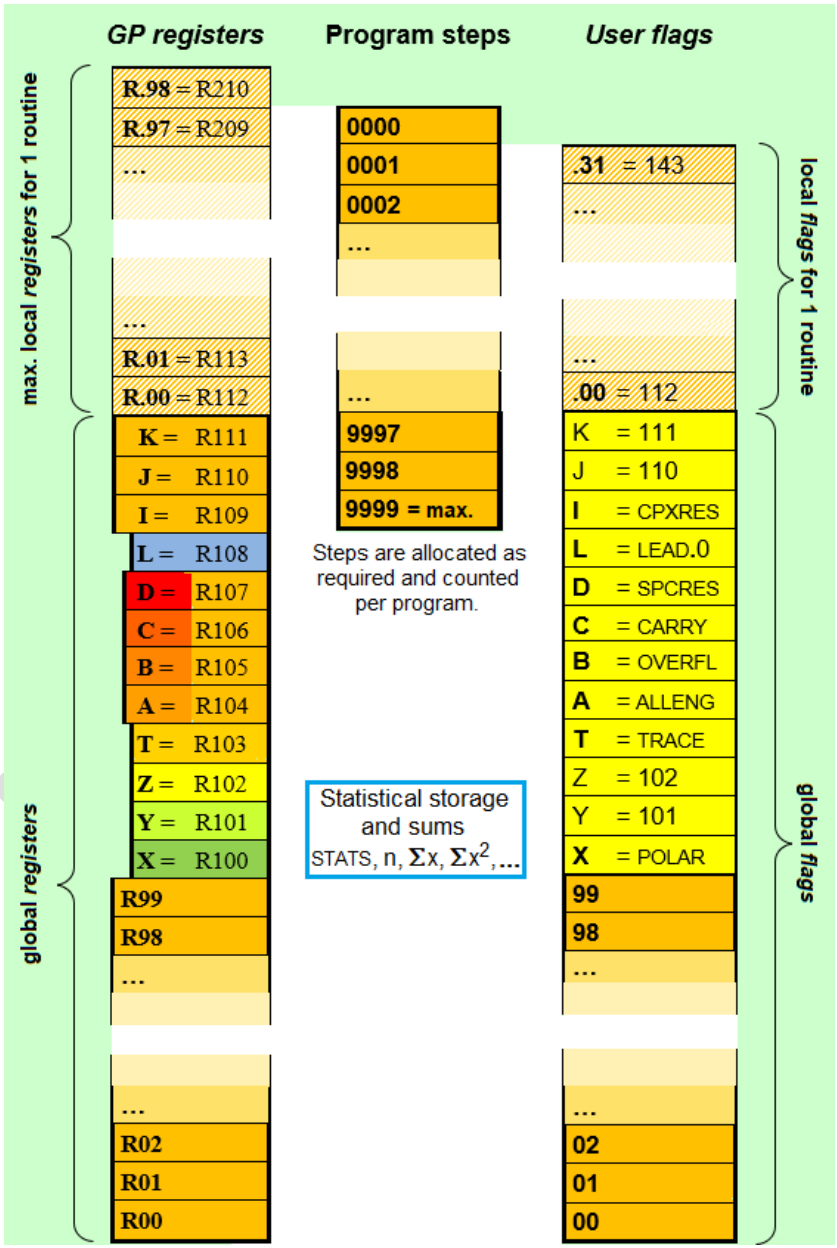
EXP 40, 142
ln 54
R↓ 80
RCL 73
STO 87
TIMER (⏱) 90
TRI 91, 146
UNDO (↶) 92
x≷y 97
XEQ 95

DRAFT

DRAFT

WP43 QUICK REFERENCE GUIDE

ADDRESS SPACE



The *stack* is your workspace for calculations. Choose a *stack* of 4 (**X**, **Y**, **Z**, and **T**) or 8 *registers* (**X**, **Y**, **Z**, **T**, **A**, **B**, **C**, and **D**; startup default). Last ***x*** is saved in **L**. Each *stack register* (e.g., **Y**) may contain any type of data (***y***).

General purpose (GP) registers: There are 100 numbered global *GP registers* (**R00** ... **R99**). Furthermore, there are **I**, **J**, and **K** serving special purposes in matrix handling (→ Q-8), probability distributions (→ Q-10), and programming but may be used globally otherwise. Also **A** … **D** may be *GP registers* unless being part of the *stack*. Each *GP register* may contain any type of data.

STO ***nn*** stores a copy of ***x*** in **R*nn***,

RCL ***nn*** recalls a copy of ***rnn*** into **X**, and

x⇄ ***nn*** swaps ***x*** and ***rnn***.

Variables are named storage locations. Each may contain any type of data. E.g., for storing ***x*** in a *variable* named ***XYZ***, type STO α *XYZ* ENTER↑.

Names of *variables* shall be unique and ≤7 *characters* long. User-created *names* (also for *menus* and *global labels*) shall start with a letter and must not contain +, −, ×, /, :, ^, (,), =, !, ;, |, comma, or a space.

Once created, *names* (also of commands) are case sensitive but level insensitive, i.e. **Bn*** accesses **B$_n$*** (easing input in equations or searches).

Flags: There are 112 global *user flags* and 43 named *system flags* (→ Q-16).

A program consists of ≥ 3 steps: LBL with a *global label*, ≥1 action step, and END. Each program may contain subroutines (containing ≥ 3 steps each: LBL, ≥1 action step, and RTN). Subroutines may be nested. Each (sub-) routine may allocate 32 local *user flags* and ≤ 99 local *registers*. Steps are counted per program (→ Q-1 and Q-5 for more).

Available memory: INFO **MEM?** and FLAG **STATUS** show the amount of free memory. Use CLP for removing programs or DELITM for deleting *variables* to free memory no longer needed.

DATA TYPES

Long integers: Any pure numeric input (without . , E , CC , or #) is taken as a *long integer* of base 10. Such numbers may be up to 1000 digits long.

Short integers: Any pure numeric input followed by # and a *base* (2 … 16) is read as a *short integer* of the base specified. D and H are shortcuts for bases 10 and 16. A *short integer* may occupy ≤ 64 bits (→ Q-14).

Real numbers: Any number entered using one . and/or E is taken as a *real number*. Each may carry 34 digits precision.

Complex numbers: A *complex number* consists of two *real numbers* combined:

DRAFT

1) its *real* and imaginary part like 1.23-i×4.56 in rectangular notation (CF POLAR (or CF X) and press **1.23** CC **4.56** +/- ENTER↑) or
2) its magnitude and phase like 7.89∡120° in polar notation (SF POLAR (or SF X) and press **7.89** CC **120** ENTER↑).

Angles: Any *real* input followed by d.ms is interpreted as an *angle* in *sexagesimal degrees*. *Angles* may be entered and displayed also in *decimal degrees*, *radians*, *multiples of* π, *mils*, or *grades* (*gon*). Choose the appropriate *angular display mode* via MODE (see overleaf).

Times: Any *real* input followed by h.ms is taken as a sexagesimal *time*. It will be displayed like 23:45:43.2109 with as many *hours* and decimals of *seconds* as chosen.

Dates: Any *real* input followed by .d is read as a *date* in the format chosen (yyyy.mmdd for YMD or dd.mmyyyy for DMY or mm.ddyyyy for MDY). It will be displayed like 2021-12-02 or 02.12.2021 or 12/02/2021, respectively. A time of day may be appended, and *times* may be added to *dates*.

Vectors and ***matrices*** → Q-8.

Text strings: α opens *alpha input mode* (*AIM*). Data entered in *AIM* become a *text string* when closed (unless they are function parameters). Latin letters A…Z are on the keyboard, more are in f A. Greek letters are accessed via g plus the corresponding Latin letter (see calculator backside). Turn to lower case by ▼ and to upper by ▲ for all letters.

f plus one of the keys +, −, ×, /, +/-, e^x, $\sqrt{x}$, ., and 0 … 9 will enter the respective *character*. Further symbols are in g − and g ..

f + R↓ makes the subsequent *character* entered a subscript, and

f + E makes it a superscript, if applicable.

Equations are entered in EQN. They may be labelled, follow algebraic notation, and may contain =, (,), ^, and a choice of monadic and dyadic functions. Such equations may be solved automatically for any *variable*, integrated, and differentiated (→ Q-12).

USING MENUS

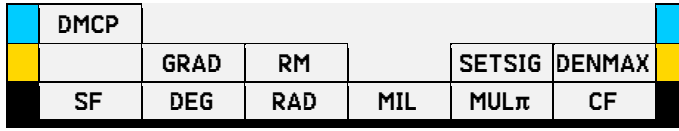
A *menu* defines the functionalities of the keys F1 – F6 by displaying up to three *labels* above each such key. To access a *label* in the lowest row, press the corresponding key. To access a *label* in 2nd or 3rd row, press f or g before. A dashed line on top of the current *menu view* indicates a *multi-view menu*; then ▲ or ▼ will show additional *views*. *Menu names* are printed underlined in running text.

Each *menu view* stays displayed until EXIT ed or another *menu* (*view*) is called.

DRAFT

MODES

MODE



SF ***n***, CF ***n*** Sets (or clears) the flag specified.

DEG Selects decimal *degrees*
RAD Selects *radians*
MULπ Selects *multiples of π*
MIL Selects *mils*
GRAD Selects *grades*, a.k.a. *gon*,
} as angular display mode.

RM ***n*** Sets rounding mode.
SETSIG ***n*** Sets calculator precision (1 ... 34 significant digits).
DENMAX ***n*** Sets the maximum denominator for calculating with fractions.

DMCP Exits *WP43* to the *DMCP menu* for file transfer or firmware update.

DISPLAY FORMATS

DISP

FIX ***n*** Fixed point display of ***n*** decimals.
SCI ***n***, ENG ***n*** Scientific (or engineering) format, SCI with ***n*** decimals.
ALL ***n*** Displays all digits as far as possible with max. ***n*** decimal zeros.

ROUND Rounds a *time*, *real*, or *complex* ***x*** to the current display format.
ROUNDI Rounds ***x*** to next *integer*.
RDP ***n*** Rounds ***x*** to ***n*** decimal places (1 ... 99, think of FIX).
RSD ***n*** Rounds ***x*** to ***n*** significant digits (1 ... 34, think of SCI).

SDL ***n***, SDR ***n*** Shifts digits left (right) in ***x*** by ***n*** decimal positions.

CHINA, EUROPE, INDIA, JAPAN, UK, USA Set local display preferences.

GAP ***n*** Selects a digit group gap inserted after every ***n*** digits.

HIDE ***n*** Set to hide all results $< 10^{-n}$, i.e. display them as 'zero'.
RANGE ***n*** Sets the maximum exponent to be displayed for *real numbers*.
HIDE?, RANGE? Returns the current HIDE or RANGE setting.

DSTACK ***n*** Sets the number of *stack registers* to be displayed (1 ... 4).

DRAFT

PROGRAMMING

Program Entry and Navigation

P/R	toggles *program entry mode*.
GTO . .	moves the *program pointer* to free program space.
GTO ▲	moves it to top of *current program* or to previous program.
GTO ▼	moves it to top of next program.
GTO . ***nnnn***	moves it to step number ***nnnn*** within the *current program*.
≡▲	moves it to previous step (you can use ▲ or ▼ unless
≡▼	moves it to next step a *multi-view menu* is displayed).
←	deletes the *current program step* entirely.
EXIT	exits *program entry mode* unless a *menu* is displayed.

Labels

A program *label* is a tag used to identify a program or a section within a program. Each program must begin with a *global label* (→ Q-2).

Global labels can be accessed from anywhere in memory (thus, they should be unique). *Global labels* are alphanumeric and ≤ 7 *characters* long.

Local labels can be accessed only within the current program (thus, they should be unique within this program). *Local labels* are A … E or numeric (00 ... 99).

Local registers

LOCR ***n*** allocates ***n*** local *registers* (≤99). 32 *local flags* come with them. Local data are valid in *current (sub-) routine* only.

Tests (Do if True, Skip if False)

When a binary test step is executed and the result is **true** then the step immediately following will be executed ("do if true"); if the test result is **false**, the step following the test step will be skipped.

```
  ...
  1.05203
  STO 'Count'
LBL 01
  ...   { loop }
  ISG 'Count'
  GTO 01
  BEEP
  ...
```

Looping

ISE, ISG, ISZ, DSE, DSL, and DSZ (in LOOP) control looping. ISE, ISG, DSE, and DSL access a *variable* or *register* containing a *loop control number* in the format ccccc.fffii with ccccc = current counter value, fff = final value, and ii = increment (or decrement) step size (default is 1). DSZ and ISZ count to 0 in steps of 1. As long as the count is not complete, the step following the instruction is executed (often a branch to the top of the loop); see also the rear side of your *WP43*. The example above counts from 1 to 52 by 3 (executing the loop 18 times) and then beeps.

DRAFT

Using a Variable Menu

A *menu* of *variables* may be displayed by the *Solver* or *Integrator* (→ Q-12f) or by VARMNU within a program. Each *label* in this *menu* represents a *variable*. While this *menu* is displayed, you can…

… store a value into a *variable*: Key in the value, then press the resp. *softkey*.

… recall the contents of a *variable*: Press [RCL], then the respective *softkey*.

… view the contents of a *variable* without recalling: Press [VIEW], then the *softkey*.

… select a *variable*: Press the respective *softkey* without typing a number first (for the *Solver*, this is how you select the unknown *variable*; for the *Integrator*, this is how you select the *variable of integration*).

You can call and use any function *menu* without exiting from the *variable menu*.

EXECUTING FUNCTIONS AND PROGRAMS

Every function or program can be executed via [XEQ] [α] ***name*** [ENTER↑] where ***name*** is the function *name* or the program *label*. Case matters. If ***name*** is not unique, the *global label* closest to the permanent end (.END.) has precedence. If ***name*** is a *local label*, *WP43* searches for it in the current program only.

Smart program menu: [XEQ] [PROG] displays all programs (actually: all *global labels*) defined at execution time. Specify the required program by pressing the corresponding *softkey*.

Single stepping: To execute the current program step, press [≡▽] (or [▼] if no *multi-view menu* is displayed). Press [≡△] (or [▲]) for browsing backwards.

Run/Stop: Press [R/S] to run the current program starting with the current step or to stop the running program after the current step is executed.

Catalog of functions: Browse [CATALOG] [FCNS] and execute the required function by pressing the corresponding *softkey*. It can be searched alphabetically.

Specifying Function Parameters

Numeric parameters: Functions accepting trailing numeric parameters will prompt you with an underscore for each digit expected. Just enter the digits; terminate shorter input using [ENTER↑]. Alphanumeric parameters are accepted by many functions as well.

***Registers* and *stack*:** Rather than keying in an actual parameter, you can specify the *variable* or *register* containing the parameter. Any function accepting a *GP register* will accept a *stack register* as well. Press the corresponding *softkey* for **X ... T** or one of the keys **A ... D**, if applicable.

DRAFT

If the wanted parameter is a *variable* already existing, pick it from VAR. If a destination doesn't exist yet, type it (e.g., for creating a *variable* ***ABC*** type STO α *ABC* ENTER↑).

Indirect addressing: Instead of keying in an actual *variable/register* address, you can specify the *variable* or *register* containing its address – just press the *softkey* →. E.g., to display the contents of the *variable* or *register* specified in **R12**, key in VIEW → **12** . This works with lettered and *stack registers* as well.

For every command ***XY*** requiring one trailing parameter, ***XY*** → X will let ***XY*** take its parameter from **X** instead.

CLEARING AND DELETING

CLR

RESET		CLKEYS	DelITM	CLMENU	
CLall	CLBKUP	DelVall		DelMall	CLLCD
CLREGS	CLPall	CLVall	CLFall	CLMall	CLSTK
CLΣ	CLP	CLCVAR	CF	CLMy...	CLX

CLΣ Clears all statistical data (also accessible via FIT and STAT).
CLP Clears (deletes) the *current program*.
CLCVAR Clears (resets) all *variables* used in the *current program*.
CF ***n*** Clears *flag* ***n***.
CLMy… *Submenu* to clear (reset) MyMENU, MyPFN, or Myα.
CLX Clears *stack register* **X**.

CLREGS Clears all *registers* (except the *stack* and statistical data).
CLPall Clears (deletes) all programs in *RAM*.
CLVall, CLMall Clears all user *variables* or *menus.*
CLFall Clears all numbered user *flags*.
CLSTK Clears the entire *stack* (i.e. resets all its *registers* to 0.).

CLall Clears almost everything but the modes set.
CLBKUP Clears (deletes) the backup file.
DelVall, DelMall Delete all user *variables* or *menus.*
CLLCD Clears the *LCD* above and to the right of pixel (***x***; ***y***).

RESET Resets your WP43 to *startup default configuration* (*SDC*).
CLKEYS Clears (deletes) all key assignments.
DelITM Deletes a user-created *item*. Please see overleaf.
CLMENU Clears the *programmable menu*.

Operations clearing programs or deleting other *items* will free space in memory.

DRAFT

DelITM VARS … ▢ Deletes the user *variable* selected.
DelITM MENUS … ▢ Deletes the user *menu* selected.
DelITM PROGS … ▢ Deletes the user program selected.

Use the applicable *softkey*

MATRIX OPERATIONS

A ***m*×*n*** *matrix* is a rectangular array featuring ***m*** rows and ***n*** columns of *real* or *complex numbers* (a.k.a. *matrix elements*).

MATX

	CMIN	CMAX	M.QR	FIND	EIGVAL	EIGVEC
	RNORM	RSUM	M.LU	DIM?		R⇄R
	I+	I-	STOIJ	RCLIJ	J-	J+
	ENORM	V∢	STOEL	RCLEL	PUTM	GETM
	dot	cross	UNITV	DIM	INDEX	EDITN
	NEW	$[M]^{-1}$	|M|	$[M]^T$	SIM EQ	EDIT

NEW Creates a new matrix in $\mathbf{X}$ (→ Q-9).
$[M]^{-1}$ Inverts the matrix $\boldsymbol{x}$.
|M| Returns the determinant of the matrix $\boldsymbol{x}$.
$[M]^T$ Transposes the matrix $\boldsymbol{x}$.
SIM EQ ***n*** Solves a system of ***n*** simultaneous linear equations (→ Q-9).
EDIT Edits the matrix $\boldsymbol{x}$ (→ Q-9).

dot Returns the dot product of the matrices (or vectors) $\boldsymbol{y}$ and $\boldsymbol{x}$.
cross Returns the cross product of the vectors $\boldsymbol{y}$ and $\boldsymbol{x}$.
UNITV Returns the unit vector of $\boldsymbol{x}$.
DIM Creates a new named matrix (→ Q-9).
INDEX, EDITN Indexes (or edits) a named matrix (→ Q-9).

ENORM Returns the *Euclidean* norm of the matrix $\boldsymbol{x}$.
V∢ Computes the *angle* between two vectors $\boldsymbol{y}$ and $\boldsymbol{x}$.
STOEL Stores $\boldsymbol{x}$ into the current element $\boldsymbol{a_{ij}}$ of the indexed matrix (*IM*).
RCLEL Recalls the current element $\boldsymbol{a_{ij}}$ of the *IM* into $\mathbf{X}$.
PUTM Puts the matrix $\boldsymbol{x}$ into the *IM*, starting with target element $\boldsymbol{a_{ij}}$.
GETM Gets a IP($\boldsymbol{y}$) × IP($\boldsymbol{x}$) sub-matrix out of the *IM* into $\mathbf{X}$, starting with $\boldsymbol{a_{ij}}$.

I+, I−, J+, J− Increments or decrements the row or column pointer for the *IM*.
RCLIJ Recalls the values of the row and column pointers into $\mathbf{Y}$ and $\mathbf{X}$.
STOIJ Sets the row pointer $\boldsymbol{i}$ to IP($\boldsymbol{y}$) and the column pointer $\boldsymbol{j}$ to IP($\boldsymbol{x}$).
R⇄R Swaps row IP($\boldsymbol{x}$) and row IP($\boldsymbol{y}$) of the *IM*.
CMIN, CMAX Returns the smallest (greatest) element in a column of the *IM*.
FIND Searches the specified value in the *IM*.

DRAFT

RNORM, RSUM Returns the row norm (or row sum) of the matrix ***x***.
M.LU, M.QR Computes the *LU* (or *QR*) *decomposition* of the matrix ***x***.
EIGVAL, EIGVEC Computes the eigenvalues (or eigenvectors) of the matrix ***x***.
DIM? Returns the dimensions of the matrix ***x*** (rows in **Y**, columns in **X**).

To create a new ***m*** × ***n*** matrix, enter its dimensions (***m*** ENTER↑ ***n***), then press

- MATX NEW for a matrix in **X** or
- MATX DIM ***r*** for a matrix in the *register* or *variable* specified or
- MATX DIM α ***name*** ENTER↑ for a matrix in the *variable* specified. If the target *variable* already exists, DIM will re-dimension it.

To edit the matrix in **X**, use MATX EDIT to start the *Matrix Editor* (*ME*).

To edit a named matrix, use MATX EDITN VAR and press the resp. *softkey*.

When a matrix is being edited it is called *indexed* (to index a matrix without editing, use INDEX). Whenever there is an indexed matrix (*IM*), two pointers (in **I** and **J**) indicate the row and column of the current element (→ Q-8).

If ***i*** and ***j*** are pointing to the last (bottom right) element in the *IM* and you press → then ...

- the pointers wrap around to the first element of the *IM* (if in *wrap mode* which is set automatically whenever you enter or exit the *ME*) or ...
- the *IM* grows by one row and the pointers move to the first element in the new row (if in *grow mode* – set GROW via the *ME menu*; reset by WRAP).

Matrix Arithmetic

+, −, ×, and / work for matrices like they do for numbers if the matrix dimensions match. Instead, advanced functions (like e^x, ln, 10^x, lg, $1/x$, $\sqrt{x}$, x^2, trigonometric and hyperbolic functions) operate on all matrix elements individually (→ *OMS 2* and the *IOI* for more).

Whenever a matrix is involved in a mathematical operation with a *complex* object, the result will be a *complex matrix*.

Solving a system of linear equations $[A]\,\vec{X} = \vec{B}$

1. Enter MATX SIM EQ ***n*** with ***n*** being the number of unknowns. Your *WP43* will create or re-dimension the matrix *variables* **Mat.A** and **Mat.B**.
2. Press Mat A; fill the matrix; eventually press EXIT.
3. Press Mat B; fill the vector; eventually press EXIT.
4. Press Mat X to compute and return the solution vector **Mat.X**.

PROBABILITY

PROB

RAN#	SEED	RANI#		lnΓ	Γ(x)
	NBin:	Geom:	Hyper:	Binom:	Poiss:
LgNrm:	Cauch:		Expon:	Logis:	Weibl:
Norml:	t:	C_{yx}	P_{yx}	F:	χ^2:

Submenu e.g.

Norml:

$\mathbf{Norml_p}$		$\mathbf{Norml_{\triangle}}$	$\mathbf{Norml_{\triangle}}$		$\mathbf{Norml^{-1}}$

$\mathbf{Norml_p}$ denotes the *probability density function*, $\mathbf{Norml_{\triangle}}$ the *cumulated distribution function*, $\mathbf{Norml_{\triangle}}$ the *error probability*, and $\mathbf{Norml^{-1}}$ the *quantile function* (this naming scheme holds also for the other *distributions,* except χ^2).

C_{yx} , P_{yx} Returns the number of possible combinations (or permutations a.k.a. arrangements) of x objects taken out of a set of y objects.

RAN# Returns a pseudo random *real* ∈ [**0**; **1**).
SEED Stores a seed ∈ [**0**; **1**) for RAN#. SEED **0** takes it from the clock.
RANI# Returns a pseudo random *integer* ∈ [IP(x); IP(y)].
Γ(x), lnΓ Returns the value of the *gamma function* (or its natural logarithm).

5 discrete and 9 continuous distributions (*d.*) are provided:

Binom:	*Binomial d.*	($i = p_0$ = gross probability of success, $j = n$ = sample size)
Cauch:	*Cauchy-Lorentz* (a.k.a. *Breit-Wigner*) *d.*	(i = location, j = shape)
Expon:	*Exponential d.*	(i = rate)
F:	*Fisher's F d.*	(i = degrees of freedom 1 (= dof_1), $j = dof_2$)
Geom:	*Geometric d.*	($i = p_0$)
Hyper:	*Hyperbolic d.*	($i = p_0$, $j = n$, k = batch size)
LgNrm:	*Log-normal d.*	($i = \mu$, $j = \sigma$)
Logis:	*Logistic d.*	($i = \mu$, j = scale parameter)
NBin:	*Negative Bin. d.*	($i = p_0$, $j = n$)
Norml:	*Normal d.*	($i = \mu$, $j = \sigma$. Standard *Gaussian d.*: $i = 0$, $j = 1$)
Poiss:	*Poisson d.*	($i = n \times p_0 = \lambda$ = *Poisson* parameter)
t:	*Student's t d.*	($i = dof$)
Weibl:	*Weibull d.*	($i = b$ = shape, $j = T$ = characteristic lifetime)
χ^2:	*Chi-square d.*	($i = dof$)

Store the required parameters in $\mathbf{I}$, $\mathbf{J}$, and $\mathbf{K}$ as specified above; the remaining parameter must be given in $\mathbf{X}$ before calling the respective function – note *quantile functions* require a probability input (**0.** ≤ x ≤ **1.**).

STATISTICS

Statistical data are collected in the ***n*×2** *real* matrix STATS, separate from the user-generated matrices.

Clear STATS via [STAT] **CLΣ** (or [FIT] **CLΣ**) before doing a new statistical analysis.

Accumulate the data:

- for each individual data value: ***x-value*** **Σ+** .
- for each weighted data value: ***weight-value*** [ENTER↑] ***x-value*** **Σ+** .
- for each x-y data pair or point: ***y-value*** [ENTER↑] ***x-value*** **Σ+** .
- For x-y data pairs stored in a two-column matrix (***x-values*** in column 1, ***y-values*** in column 2), put this matrix in **X** and then press **Σ+** .

To undo input errors or remove erroneous data,

- use **Σ-** or [↶] to remove the last data point entered from STATS or…
- edit STATS using the *Matrix Editor* (→ Q-8).

Data Analysis

		$\bar{x}_H$				
		$\bar{x}_{RMS}$	x_{min}	x_{max}	Δx	
		MEDIAN	MAD	Q_3-Q_1	%ile	
[STAT]	CLΣ	$\bar{x}_G$	ε	ε_p	ε_m	FIT
	Σ-	$\bar{x}_w$	s_w	σ_w	s_{mw}	HISTOG
	Σ+	$\bar{x}$	s	σ	s_m	SUM

SUM — Recalls Σy and Σx.

$\bar{x}$, s, σ, s_m — Arithmetic mean value, sample standard deviation (*SD*), population *SD*, standard error (a.k.a. *SD* of the mean), all for x and y.

$\bar{x}_w$, s_w, σ_w, s_{mw} — Same for weighted data.

HISTOG — Submenu to prepare and plot a histogram (→ *OMS 2*).

$\bar{x}_G$, ε, ε_p, ε_m — Geometric mean value, sample scattering factor (*SF*), population *SF*, *SF* of the mean, all for x and y.

FIT — See overleaf.

MEDIAN, MAD — Median and median absolute deviation.

%ile, Q_3−Q_1 — Percentile, distance between 75%- and 25%-percentiles.

x_{min}, x_{max}, Δx — Min. and max. values in the data set and their difference.

$\bar{x}_H$, $\bar{x}_{RMS}$ — Harmonic and quadratic means.

DRAFT

	GaussF	CauchF		BestF		
	ParabF	HypF	RootF			
	LinF	ExpF	LogF	PowerF		OrthoF
FIT	CLΣ	ASSESS				PLOT
	Σ-		s_{xy}	cov		HISTOG
	Σ+	L.R.	s(a)	r	$\hat{x}$	$\hat{y}$

L.R. Computes the parameters a_0 and a_1 (and a_2, if applicable) of the regression for the fit model (*FM*) selected (see below).

s(a) Returns the standard errors of a_0 and a_1 for the green *FM's*.

r Returns the correlation coefficient between {x} and {y}.

$\hat{x}$, $\hat{y}$ Returns the forecast for x (or y) according to the *FM* selected.

s_{xy}, cov Returns the sample (or population) covariance.

ASSESS Plots the data points together with the *FM* curve(s) selected.

- CauchF *Cauchy* peak *FM*: $y = 1/[a_0\,(x + a_1)^2 + a_2]$.
- ExpF Exponential *FM*: $ln(y) = ln(a_0) + a_1 x$ or $y = a_0 e^{a_1 x}$.
- GaussF *Gauss* peak *FM*: $y = a_0\, e^{\frac{(x-a_1)^2}{a_2}}$.
- HypF Hyperbolic *FM*: $y = 1/(a_0 + a_1 x)$.
- LinF Linear *FM*: $y = a_0 + a_1 x$.
- LogF Logarithmic *FM*: $y = a_0 + a_1 ln(x)$.
- ParabF Parabolic *FM*: $y = a_0 + a_1 x + a_2 x^2$.
- PowerF Power *FM*: $ln(y) = ln(a_0) + a_1 ln(x)$ or $y = a_0\, x^{a_1}$.
- RootF Root *FM*: $y = a_0\, a_1^{1/x}$.

BestF ***n*** Picks the *FM* returning the best r^2 of all *FM*s allowed (***n*** = *FM* code).

OrthoF Works like LINF but assumes equal errors in x and y. Note ORTHOF is not part of the set of *FM's* which BESTF checks.

PLOT Plots measured data points to analyse the measuring system under investigation; with ≥30 points, its standard error can be determined.

ADVANCED OPERATIONS

EQN is for interactive editing, storing, solving, integrating, and differentiating,

ADV for programmed summing, multiplying, solving, integrating, differentiating.

Interactive Operations on Equations

EQN

...					
DELETE					
NEW	EDIT	f''	f'	∫f	Solver

DRAFT

For creating a new equation, press NEW. The *Equation Editor menu* will open, and the blue row will display the *current equation*. Press EXIT when finished.

For browsing existing equations, press ▲ or ▼. The equation displayed in blue row is called the *current equation*.

For editing (or deleting) the *current equation*, press EDIT (or DELETE).

For operating on the *current equation*, press the respective *softkey*. A *menu* will pop up displaying the *names* of all *variables* used therein (maybe more).

Using Advanced Operations in Programs

ADV

	PGMSLV	SLVC	f"(x)			PGMINT
	SOLVE	SLVQ	f'(x)	Π_n	Σ_n	∫fdx

SLVQ (or **SLVC**) solves the quadratic (or cubic) equation $ax^2 + bx + c = 0$ (or $ax^3 + bx^2 + cx + d = 0$) with all its parameters on the input *stack* [(***d***,) ***c***, ***b***, ***a***, …]. SLVQ will return two *real* or two *conjugate complex* roots; SLVC will return three roots and the *discriminant*.

Π_n ***label*** calculates the product …

Σ_n ***label*** calculates the sum …

} … of the terms given by the routine specified, using the loop control number in **X** (→ Q-5).

SOLVE ***var*** solves for an unknown *variable* in an expression ***f(x₁, x₂, ...)* = 0** when values for all the other *variables* are given. The expression shall be written as a program (call it **AB**, for example):

- **AB** must begin with a *global label*.
- The body of **AB** shall evaluate the expression. For an expression to be solved, it must be coded that ***f(x₁, x₂, ...)* = 0** is fulfilled. Recall the *variables* of the expression as they are required and calculate ***f***.
- **AB** must logically end with RTN.

Then write a 2nd program (call it **CD**, for example) calling the *Solver*. At the position where you need the expression solved, press ADV, then…

1. Press PGMSLV and specify **AB**.
2. Store a value into each known *variable*, e.g., using STO. Optionally store a guess into the unknown *variable* to direct the *Solver* to a solution.
3. Press SOLVE and specify the unknown *variable*.

When running **CD** later on, SOLVE will solve for the unknown.

f'(x) (or **f"(x)**) calculates the 1st (or 2nd) derivative of ***f(x)*** at location ***x***. A program (e.g., called **EF**) shall evaluate ***f(x)***; it must begin with a *global label* and take care of all *variables* used.

DRAFT

Then write a 2nd program calling the derivation (call it **GH**, for example).

1. Store a value into each *variable* to remain constant under derivation.
2. At the position where you need the derivative, put the respective location into X, then press ADV f'(x) (or f"(x)) specifying **EF**.

When running **GH** later on, the derivative will be returned in X.

∫fd *var* computes a definite integral. The integrand ***f(x)*** shall be written as a program (e.g., called **IJ**); it must begin with a *global label*, recall all integration constants used, and evaluate ***f(x)***.

Then write a 2nd program (call it **KL**, for example) calling the *Integrator*. At the position where you need the integral, press ADV, then…

1. Press PGMINT and specify **IJ**.
2. Press ∫fdx . A *submenu* will open.
3. Store a value into each *variable* that shall remain constant under integration, e.g., using STO.
4. Store the lower (↓LIM) and upper limit (↑LIM) of integration, and the ACCuracy factor.
5. Press ∫ and specify the *variable of integration*.

When running **KL** later on, it will return the integral in X and its uncertainty in Y.

OPERATIONS ON SHORT INTEGERS

		→BIN	→OCT			→DEC	→HEX
		1COMPL	2COMPL	UNSIGN	SIGNMT		WSIZE
INTS		DBL/	DBLR	DBL×	^MOD	ceil	GCD
		IDIV	RMD	MOD	×MOD	floor	LCM
		A	B	C	D	E	F

A ... **F** Digits for input of *short integers* of bases >10.

IDIV Integer divide – works also for *reals* (ℝ) and *long integers* (ℤ).

RMD (ℝ, ℤ) Remainder.
MOD (ℝ, ℤ) Modulo.
×MOD (ℝ, ℤ) Returns $(z \cdot y) \bmod x$.
^MOD (ℝ, ℤ) Returns $(z^y) \bmod x$.

floor, ceil (ℝ) Returns the greatest *long integer* ≤ x (or the smallest ≥ x).

LCM, GCD (ℤ) Returns the least common multiple (or the greatest common divisor) of x and y.

DBL/, DBLR, DBL× Double *word* length commands for division, remainder, and multiplication.

DRAFT

1COMPL, 2COMPL Sets 1's (or 2's) complement mode.
UNSIGN Sets unsigned mode.
SIGNMT Sets sign-and-mantissa mode.
WSIZE ***n*** Sets the *word* size to 1 ... 64 *bits*.
→BIN, →OCT, →DEC, →HEX Convert the display of ***x*** directly.

BIT

	1COMPL	2COMPL	UNSIGN	SIGNMT		WSIZE
	LJ				RJ	ASR
	SL	RL	RLC	RRC	RR	SR
	A	B	C	D	E	F
	SB	BS?	#B	FB	BC?	CB
	NAND	NOR	XNOR	MIRROR		
	AND	OR	XOR	NOT	MASKL	MASKR

AND, OR, XOR, NAND, NOR, XNOR *Boole's* binary operators.
NOT Inverts every bit in ***x***.
MASKL, MASKR ***n*** Creates a mask of ***n*** bits on the left (or right) side.
MIRROR Reflects all bits.

SB, FB, CB ***n*** Sets, flips, or clears bit #***n*** in ***x*** (counting starts with #**0**).
BS?, BC? ***n*** Checks if bit #***n*** in ***x*** is set (or clear).
#B Returns the total number of bits set in ***x***.

SL, SR ***n*** Shifts ***x*** left (or right) by ***n*** places.
ASR ***n*** Arithmetically shifts ***x*** to the right by ***n*** places, i.e. divides ***x*** by 2^n.
RL, RR ***n*** Rotates ***x*** left (or right) by ***n*** places.
RLC, RRC ***n*** Rotates ***x*** left (or right) by ***n*** places through CARRY.
LJ, RJ Adjusts the bits set in ***x*** to the left (or right).

OPERATIONS ON TEXT STRINGS

Use [+] to connect strings. Then ***x*** will be appended to ***y***. With numeric data in **X** or **Y**, their current display format is taken into account.

α.FN

FBR					
				αLENG?	αPOS?
x→α	αRL	αRR	αSL	αSR	α→x

x→α ***d*** Converts the code ***x*** to the corresponding *character* and appends it to the string in ***d***.

αRL, αRR ***s*** Rotates the string in ***s*** by ***x*** *characters* to the left (or right).
αSL, αSR ***s*** Deletes the first (or last) ***x*** *characters* of the string in ***s***.

DRAFT

α→x ***s*** Removes the 1st *character* from ***s*** and pushes its code on the *stack*.

αLENG? ***s*** Pushes the length of the string in ***s*** on the *stack*.

αPOS? ***s*** Returns the position where substring ***x*** begins in the string ***s***.

FBR Shows all *characters* defined in both fonts.

USER WRITEABLE SYSTEM FLAGS

The *startup default* state of each *flag* is indicated (~~FLAG~~ means 'FLAG clear'). STATUS shows all *flags* currently set, some *flags* appear also in the *status bar*.

~~ALLENG~~ (A)	ALL overflows to ENG, else to SCI
~~ALPHA~~	*WP43* in *AIM*
~~ALP.IN~~	Alphanumeric input received, else NUM.IN
AUTOFF	Turns *WP43* off after 10 minutes inactivity, else stays on
~~AUTXEQ~~	Executes current program automatically when *WP43* is turned on
~~CARRY~~ (C)	For *short integer* manipulation
~~CPXRES~~ (I)	Allows *complex* results for *real* input
DECIM.	Displays decimal points, else commas
DENANY	Allows for any denominator up to D.MAX
~~DENFIX~~	Fixed denominator = D.MAX, else factors of D.MAX are allowed
~~FASTFN~~	Faster function computation
~~FRACT~~	Set for fraction display, clear for decimal display
~~FRCSRN~~	Verbose fraction display showing *register names*
~~IGN1ER~~	Ignores one error
~~LEAD.0~~ (L)	Displays leading zeros for *short integers*
MULT×	Displays multiplication crosses, else points
~~NUM.IN~~	Numeric input received, else ALP.IN
~~OVERFL~~ (B)	For *short integer* manipulation
~~POLAR~~ (X)	Polar notation for *complex numbers*, else rectangular
PROPFR	For FRACT set: displays proper fractions, else improper
~~QUIET~~	Disables the beeper
~~SPCRES~~ (D)	Allows for special *real* results, i.e. ±∞ and NaN
SSIZE8	Sets *stack* size to 8 *registers*, else 4
TDM24	Sets *time* display to 24 hours, else 12
~~TRACE~~ (T)	Print line is in tracing mode
~~USER~~	User mode

Copyright © 2017 - 2025 Walter Bonin, Radebeul, Germany

BACKGROUND FACTS AND CONSIDERATIONS

This section is for recording and explaining some of the boundary conditions considered and settings chosen for the *WP43* in the course of this project. It is not necessary for operating the *WP43* but may foster understanding, and also a bit of the product philosophy may be found here.

WP43 is a Gesamtkunstwerk. Creating it, our primary goals were and are

1. Coherence,
2. Internationality (using international symbols instead of national abbreviations wherever possible (e.g., [+/−] instead of [CHS]), complying to *ISO* standards, etc.),
3. Reliability, and
4. Accuracy, in this order.

Note there are no update marks in the following text since I presume you will read entirely what you are interested in.

With a project developed in the public for over a decade, some of our ideas diffused elsewhere. E.g., named *summation registers* (actually invented with the *WP 34S*) are now found in *Plus42* as well.

DRAFT

Accessing Items

The calculator hardware offers 43 keys for over 700 *functions* plus 79 constants. Subtract six for the *menu* keys. Thus, space for primary functions is quickly occupied – the respective functions are mostly set:

Obvious primary functionalities are the digits [0] … [9], [.], [ENTER↑], [x⇄y], [+/−], [E], [⌫], [+], [−], [×], [/], [STO], [RCL], [f] and [g], taking 24 key tops. [▲], [▼], and [EXIT] are required for menu handling. [XEQ] and [R/S] come with programming. So only eight locations are left. Once you want to deal with *complex numbers* seriously, you will need a primary [CC].

Other functions may be debated: [R↓], [1/x], [y^x], [x^2], [$\sqrt{x}$], [e^x], [ln], [10^x], [lg], [sin], [cos], and [tan] are the most popular – 12 candidates for seven key tops remaining. Either [x^2] or [$\sqrt{x}$] shall be primary; and we can ditch [10^x] and [lg] when we have [e^x] and [ln] primary. So the six functions [R↓], [1/x], [y^x], [sin], [cos], and [tan] compete for the remaining four key tops. We chose the first two and put the latter four (and their inverses) under two primary *menus*: [EXP] and [TRI] let you access each of y^x, $\log_x y$, 2^x, $\mathrm{lb}\,x$, x^2, x^3, sin, cos, tan, arcsin, arccos, and arctan with two keystrokes maximum (no more than what you need if they were shifted labels on the keyboard).

[10^x] and [lg] shall become secondary functions. But is it better to have them as shifted keyboard functions or as unshifted *softkeys*? No definite answer can be given here since it depends on the time the respective *menu* will stay on screen. We made [10^x] and [lg] both [f]-shifted functions since they may be missed if not seen immediately on the keyboard. (For similar reasons, [x≷], [SF], and [CF] are present on the keyboard although they are unshifted *softkeys* as well – there is no pressing need to remove them from the keyboard so far.)

WP43 features 35 *menus* on its keyboard. Of these, CATALOG, CONST, and the three alpha *character menus* shall be treated separately since they follow special rules. Each of the other 30 *menus* offers 6 unshifted locations for its most popular *items*. Selecting these can be easy (like in ADV, LOOP, STK, or TRI) or more difficult / arbitrary (like in INFO, P.FN, or X.FN). Though changing them is 'only software'.

Summarizing the labels on the keyboard and unshifted *softkeys* (cf. pp. 140ff) we get the following functions requiring up to three keystrokes now (those of them requiring trailing parameters are printed on light green):

Keystroke(s)	Menu	Functions					
1	none	0 … 9	.	E	←	▲	▼
		EXIT	ENTER↑	x≷y	R↓	+/-	+
		-	×	/	STO	RCL	XEQ
		R/S	CC	1/x	ln	e^x	$\sqrt{x}$
1 + 1	EXP	x^3	y^x	$\log_x y$	lb x	2^x	x^2
	TRI	sin	arcsin	cos	arccos	tan	arctan
2	none	a b/c	#	d.ms	.d	h.ms	α
		lg	10^x	ASN	SAVE	RBR	VIEW
		R↑	|x|	∢	SNAP	USER	FILL
		DROP↓	x≷	Δ%	DSP	↶	||
		MOD	GTO	LBL	RTN	x!	⏱
		R←	→P	BST	SST	π	SHOW
		P/R	OFF	SF	CF	USB	DMCP
2 + 1	ADV	SOLVE	SLVQ	f'(x)	Π_n	Σ_n	∫fdx
	BIT	AND	OR	XOR	NOT	MASKL	MASKR
	CLK	DATE	→DATE	DATE→	WDAY	TIME	x→DATE
	CLR	CLΣ	CLP	CLCVAR	CF	CLMy…	CLX

DRAFT

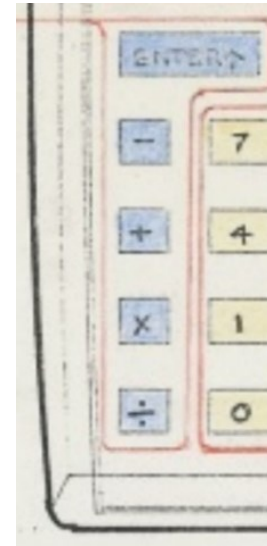
Keystroke(s)	Menu	Functions					
	CPX	dot	cross	UNITV	Re	conj	Re≷Im
	DISP	FIX	SCI	ENG	ALL	ROUNDI	ROUND
	EQN	NEW	EDIT	f''	f'	∫f	Solver
	FIN	%	%MRR	%T	%Σ	%+MG	TVM
	FIT	Σ+	L.R.	s(a)	r	$\hat{x}$	$\hat{y}$
	FLAG	SF	FS?	FF	STATUS	FC?	CF
	INFO	SSIZE?	MEM?	RM?	ISM?	WSIZE?	KTYP?
2 + 1	INTS	A	B	C	D	E	F
	I / O	LoadAll	LoadPr	LoadEq	DUMP	WRITEP	SAVEST
	LOOP	DSE	DSZ	DSL	ISE	ISZ	ISG
	MATX	NEW	$[M]^{-1}$	|M|	$[M]^T$	SIM EQ	EDIT
	MODE	SF	DEG	RAD	MIL	MULπ	CF
	PART	IP	FP	MANT	EXPT	sign	DECOMP
	PRINT	🖶x	🖶r	🖶Σ	🖶ADV	🖶LCD	🖶PROG
	PROB	Norml:	t:	C_{yx}	P_{yx}	F:	χ^2:
	P.FN	INPUT	MVAR	VarMNU	END	PAUSE	P.FN2
	STAT	Σ+	$\bar{x}$	s	σ	s_m	SUM
	STK	x≷	y≷	z≷	t≷	≷	DROPy
	TEST	x<?	x≤?	x=?	x≠?	x≥?	x>?
	U→	E:	P:	Time:	F&p:	m:	x:
	X.FN	AGM	B_n	B_n*	erf	Ellipt	Orthog
	α.FN	x→α	αRL	αRR	αSL	αSR	α→x
	Σ	n	Σx	Σx^2	Σxy	Σy^2	Σy
	∡→	→DEG	→RAD	→MULπ	→MIL	→GRAD	→D.MS
1 + 2	EXP	$\sqrt[3]{x}$	$\sqrt[x]{y}$		ln(1+x)	e^x-1	$\sqrt{(1+x^2)}$
		sinh	arsinh	cosh	arcosh	tanh	artanh
	TRI	sinc	sinc				atan2
		sinh	arsinh	cosh	arcosh	tanh	artanh

DRAFT

All the first ca. 160 *menu* functions listed above can be accessed via a single keystroke if, when, and as long as their respective *menu view* is open. Thus, repeating any unshifted *softkey* as a shifted keyboard *label* is of limited value. It is done for three functions only, as mentioned above.

After drawing many fictional calculators just for fun, I gained my real-world layout experience with the *WP 34S*. Based on this and seeing at the horizon a forthcoming better display than the very limited one of the *HP-20b/30b*, I conducted a poll about the community's preferences for placing the four basic arithmetic operators on a hypothetical *RPN* portrait pocket calculator (see *https://www.hpmuseum.org/cgi-sys/cgiwrap/hpmuseum/archv021.cgi?read=234505*) in the *MoHPC* in 2012-11. (Instead of resurrecting an old device or copying an existing layout, we wanted to create a really new, optimized pocket calculator based on our design experience – and we believed in getting our product made by an *OEM* on a contract basis as soon as we were ready.)

Said poll returned in a majority of votes for placing the arithmetic keys beneath ENTER↑ for ergonomic reasons – like on *HP's Classic* calculators, see the section of an *HP-35* design sketch here (copied from *https://calc.fjk.ch/der-legendaere-hp-35/*). The picture overleaf shows my last (internal) layout before the poll, taking the colors of *Dieter Rams' Braun ET66* (the calculator which inspired *Steve Jobs*) and wishfully assuming the good old slanted *HP* keys of the 70's. So I posted a first draft of the *WP 43S* incl. its innovative *menu* concept featuring three rows of softkeys displayed in the *menu section* (a concept I had outlined in an article for the British *HPCC Datafile V27 N3* in 2008-05 already) in return in November 2012 (see *https://www.hpmuseum.org/cgi-sys/cgiwrap/hpmuseum/archv021.cgi?read=234685*).



Since the sequence and location of the four basic arithmetic operators on the keyboard may become

DRAFT

subject to vivid discussions even after more than a decade, let's take a short look at history:

All the early *HP* pocket calculators (from the world's first scientific pocket calculator, the *HP-35* of 1972, up to the *HP-41CX* of 1983) put them beneath [ENTER↑] in the order [−], [+], [×], and [÷] (top down). Though actually no one knows the reason for this vintage arithmetic operator sequence anymore **(**we would have to ask *Tom Osborne* and hope he will remember still after more than 50 years, see *http://www.hp9825.com/html/osborne-s_story.html*:

The *HP-35* project was just plain fun. We knew it would work (…) so we spent a lot of time deciding whether the arithmetic keys should be on the right or the left, and whether the "+" key should follow the convention of adding machines and be placed in the lower right corner, or whether it should be located where it is most convenient (for whatever reason).**)**.

Launching the *Voyagers* in landscape format in 1981, the arithmetic keys had to move from left to the right of the numeric keys. In that course, *HP* abandoned its proprietary operator sequence for pocket calculators (re-) turning to the conventional adding machine scheme with [+], [−], [×], and [÷] put bottom up (which *HP* had featured on all their desktop devices anyway). *HP* kept this scheme for all their calculators following until today.

After said first post of the *WP 43S* in 2012-11, just five keytops in second and one in third row changed until today. Everything else was patient refinement and careful tuning – shifted functions wandered around (cf. pp. B-1ff); slanted keys are not available anymore.

Our calculator layout even survived an hardware switch – we were still waiting for *Eric Smith's* and *Richard Ottosen's* so-called *Nordic Reptiles* (see *US HHC* meetings until 2015) when *Michael* mailed me the concept of his and *David's DM42* in 2016-03. Development of the *DM42* easily overtook the *WP 43S* – it was launched as a resurrection of the *HP-42S* late in spring 2017 while *Paul* and I were looking for actual support (beyond friendly words) and willing software engineers still in vain. Despite the popularity of the *WP 43S* in the reading community, lack of qualified coding staff was the lasting bottleneck in our project since 2012. It was overcome not earlier than in 2021, when more than only one programmer joined our project.

After long negotiations (*SM* didn't want to just manufacture batches of the hardware for us), a *WP43* pilot batch was produced by *SM* in the summer of 22. It was sold at the *Allschwil Meeting* 2022 and in few weeks following. Thereafter, we just filled the free lots on shifted [7] and [8].

Just in case: Anyone wanting to modify the *WP43* layout for whatever reason must tell me which function he wants to replace where, and where this displaced function shall go to instead, until all functions are placed again. No other modification will be even considered.

The *C43/47* team (which forked from *WP 43S* in 2019), on the other hand, has pursued a layout featuring a single shift key for years, primarily to stay with the keytops of *DM42* unchanged. The necessary second shift is reached there by pressing said shift key twice. They say they will continue this way.

Additionally they unveiled a dual shift key layout in 2023-12, called *R47*. Therein, 28 keys match *WP43*, eight are just mirrored, and only seven keys plus (of course) the bezel differ significantly. A pilot batch has been manufactured by *SM*, and first *R47* calculators are expected to be sold soon.

DR

Angles

Originally, a separate *DT* for *angles* was planned. It was removed in v0.9 since its scope is quite limited and the opinion rose that '*angles* work like *reals*'. It turned out, however, that D.MS data would need special treatment in calculations, so *DT 4* returned with v0.10 for sake of keeping algebraic operations simple for the user and allowing for standard operations instead of dedicated commands like D.MS+, D.MS−, etc.

Actually, *angles* are displayed in six units or '*modes*' (*ADM*: *decimal* and *sexagesimal degrees*, *radians*, *mils*, *multiples of* π, and *gon* or *grades*). They were represented internally in a fixed format of 1296 units per turn – similar to *short integers* where a fixed bit pattern may be displayed differently depending on *integer sign modes* (*ISM*) and bases selected. *Radians*, however, did not fit into this paradigm due to the need for high precision storage of 2π for modulo calculations and reduction of rounding errors. And *radians* seem inevitable since the (standard) *Taylor* series for trigonometric functions are written for *radians* (although they can be easily rewritten for *multiples of* π). So, internally, *angles* are specified as tagged *reals* now (cf. App. B).

Generally, the trigonometric functions shall operate on *angles* within ±180° only; thus, numeric input beyond ±180° is

- converted to *real*, if applicable, then
- reduced modulo 360°, then
- minus 180° (or equivalents in other *ADMs*)

before converting it to *radians* and executing the respective function.

Example: cos(6381956970095103×2^{797}) = ? in RAD

Solution: **797** [2^x] **6 381 956 970 095 103** [×] returns

5 319 372 648 326 541 416 707 296 656 673 541 083 813 475 031 793
921 822 105 998 164 685 326 343 987 747 477 646 239 125 204 069
843 392 466 931 105 720 371 047 561 653 378 447 496 736 288 905
533 500 277 726 150 903 890 962 697 774 418 679 535 123 008 556
835 980 236 851 047 840 822 029 788 166 318 932 319 835 828 816
270 258 618 761 216 exactly.

We know from *WolframAlpha* that the cosine of this angle should return

−4.687 165 924 254 627 611 122 582 801 963 884 398 777 914 718 971
083 783 256 977 689 161 920 374 093 894 172 163 638 405…×10^{-19}

which means that the reduced angle has been

1.570 796 326 794 896 619 700 038 284 065 214 203 210 842… *radians*

DRAFT

which deviates from π/2 (= 1.570 796 326 794 896 619 231 321 691 639…) just by some 4.7×10^{-19} as it should.

Again, the crucial mode is *radians* (as long as these functions are basically evaluated in *radians*) since full circles can't be represented in *rad* exactly but just with 34-digit accuracy. Thus, input $\gtrsim 10^{34}$ completely loses significance in modulo 2π reduction since it can't support a defined fraction of a full circle anymore.

On the other hand, *WP 34S* has demonstrated that 451 digits for 2π suffice to warrant 16-digit accuracy of function results within the range of *single precision reals* (see https://forum.swissmicros.com/viewtopic.php?f=2&t=350#p4349) – though neglecting the significance limit mentioned above. In full analogy, *WP43* uses 1071 digits of 2π to warrant 34-digit accuracy of respective function results within $\pm10^{999}$. Thus, RANGE 999 is the maximum sensible setting for *WP43* if you don't want to confine trigonometric functions to sensible arguments.

An agreement about the right way to deal with this 'large input angles' problem could not be reached so far.

Calculation Internals

For $y > 0$, general powers in $\mathbb{R}$ and $\mathbb{C}$ are calculated as $y^x = e^{x\,ln(y)}$ and general roots as $\sqrt[x]{y} = e^{\frac{ln(y)}{x}}$ for all values of x except the *integers* 2 and 3. For odd ('integer') roots of $y < 0$, $\sqrt[x]{y} = -e^{\frac{ln(-y)}{x}}$; here, 'integer' includes *DT 1* and *10* and *reals* with zero fractional part. Some special results in App. B can be deduced from these calculation paths.

Note that *Free42* may calculate even powers differently (as a series of multiplications using repeated squaring), never employing more than 34 digits (see also p. B-20).

Character Sets

The browser FBR displays the *characters* of both fonts provided as designed and implemented for the *WP 43S* very early in this project, sorted according to their hexadecimal codes (most of them following *Unicode*). Some *characters* had to be 'invented' in either font since I found no Unicode equivalent in the respective range. Dedicated *characters* were created, e.g., for Euler's e and complex i for sake of unambiguity.

DRAFT

The so-called **'numeric' font** uses a matrix of up to 16 × 32 px (variable width, fixed height). Therein, the punctuation space (2008_{16}, 8 px wide) is employed for separating groups of digits in longer numbers, following *ISO 80000-1* for an unambiguous numeric display. This font is generally used for numeric output of the *WP43*. It is also employed for echoing numeric input unless too long. It could be used for echoing command input as well – screen space suffices.

Six blank *characters* are provided here allowing for any spacing wanted (standard / em / figure, punctuation, four-per-em, and hair space being 16, 8, 4 and 1 px wide).

Most of the elevated *characters* are for exponents or fraction numerators. The digits below are for denominators. Numeric indices are for indicating bases of *short integers*. Non-numeric indices are mainly provided for CONSTants.

Optionally, narrower digits (2470_{16} ff) can be used in *complex numbers*, matrices, or *short integers* of small base where space may be scarce (see pp. B-14ff).

All *characters* of the **standard (a.k.a. small) font** of alphanumeric *characters* as designed and implemented live in a matrix of up to 14 × 20 px (variable width, fixed height again). Herein, *characters* usually start at column one and feature two empty columns at their right end.

Characters with codes $< 0020_{16}$ are for control purposes; e.g., 4, 10_{10}, or 27_{10} may be useful for printer control (e.g., of an *HP-82240 A/B*).

Many symbols are 8 px wide as digits, helping where a constant symbol spacing is wanted. – Many super- and subscripts are also provided, allowing for displaying all the *items* featured on *WP43*. Arbitrary numeric indices or exponents are possible as well.

Eight blank *characters* are featured (listed here with their addresses and widths). Using them, any spacing is viable.

	Code	px
Standard space	20_{16}	10
m space	2003_{16}	12
$^{m}/_{3}$ space	2004_{16}	4
$^{m}/_{4}$ space	2005_{16}	3
$^{m}/_{6}$ space	2006_{16}	2
Figure space	2007_{16}	8
Punctuation sp.	2008_{16}	4
Hair space	$200a_{16}$	1

This set of small characters allows for correctly spelling the languages of >3.5×10⁹ people using either Greek or Latin alphabets:[188]

Afrikaans, aymara, Bahasa Indonesia, Bahasa Melayu / Malaysia, Basa Jawa, Basa Sunda, Bhāsa Bali, bosanski, català, Cebuano (Bisayan), čeština, Cymraeg, dansk, deutsch, eesti, Ελληνικά,

[188] See *https://de.wikipedia.org/wiki/Liste_lateinischer_Alphabete#Erweiterungen* .

DRAFT

English, español, euskara, Filipino, français, Gaeilge, galego, Hiligaynon, hrvatski, Ilokano, italiano, Kiswahili, kreyòl (ayisien), kurdî, lietuvių, magyar, Malagasy, Nāhuatl (Mexicatlatolli), nederlands, nihongo (rōmaji), norsk, Oʻzbek tili, polskì, português, quechua (runasimi), română, shqip, slovenščina, slovensky, srpski, suomi, svenska, Tagalog, tatarça, Türkçe, Türkmen dili, Vlaams, walon, Waray, and zhōngwén (hànyǔ pīnyīn).

This makes the *WP43* the most versatile multilingual calculator available worldwide. If you know of further living languages covered (with ≧ 1 million speakers) beyond the ones listed above, please tell us. Turn to the *OM* for examples where and how these characters can be used.[189]

Find below some 900 *characters* I have designed for the WP43_NumericFont and the WP43_StandardFont and their codes (note 185 *UTF* codes were abused for the *characters* printed on blue – I didn't find any equivalents for these in the *UTF* range covered):

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0020		!		#		%		'	(	)	*	+	,	-	.	/
0030	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0040		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0050	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0060		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0070	p	q	r	s	t	u	v	w	x	y	z		|		~	
...																
00b0	°	±				µ		·								
...																
00d0								×								
...																
0170								ŷ								
...																
0230			$\overline{y}$	$\bar{y}$												
...																
0370									$\bar{x}$	$\hat{x}$						$\overline{x}$
...																

[189] Some *characters* you see displayed by FBR are not found in any other *menu* of your *WP43*. They are not required for any *item* provided so far and may be for future use.

DRAFT

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0390		Α	Β	Γ	Δ	Ε	Ζ	Η	Θ	Ι	Κ	Λ	Μ	Ν	Ξ	Ο
03a0	Π	Ρ		Σ	Τ	Υ	Φ	Χ	Ψ	Ω						
03b0		α	β	γ	δ	ε	ζ	η	ϑ	ι	κ	λ	μ	ν	ξ	ο
03c0	π	ϱ		σ	τ	υ	φ	χ	ψ	ω						
...																
2000																
2010									‘	’						
...																
2080	0	1	2	3	4	5	6	7	8	9	+	−				*
2090															∞	∞
...																
2100			ℂ												h	ℏ
...																
2140								*e*	*i*							
...																
2160	0	1	2	3	4	5	6	7	8	9	+	−				
...																
21c0	→				≷											
...																
2210											√	∛	∜		∞	
2220		∡										∫				
...																
2240									≈							
...																
2260	≠				≤	≥										
...																
2290						⊕	α	δ	μ		⊙					
...																
2390										🖶						
23a0		⌈	⌉	⌊	⌋	⌈	⌋									
...																
2420								␣	␊	␋				UK	US	
...																
2460	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
2470				0	1	2	3	4	5	6	7	8	9	10		
2480								f	g	h						
2490				r						×			a	b	c	d

DRAFT

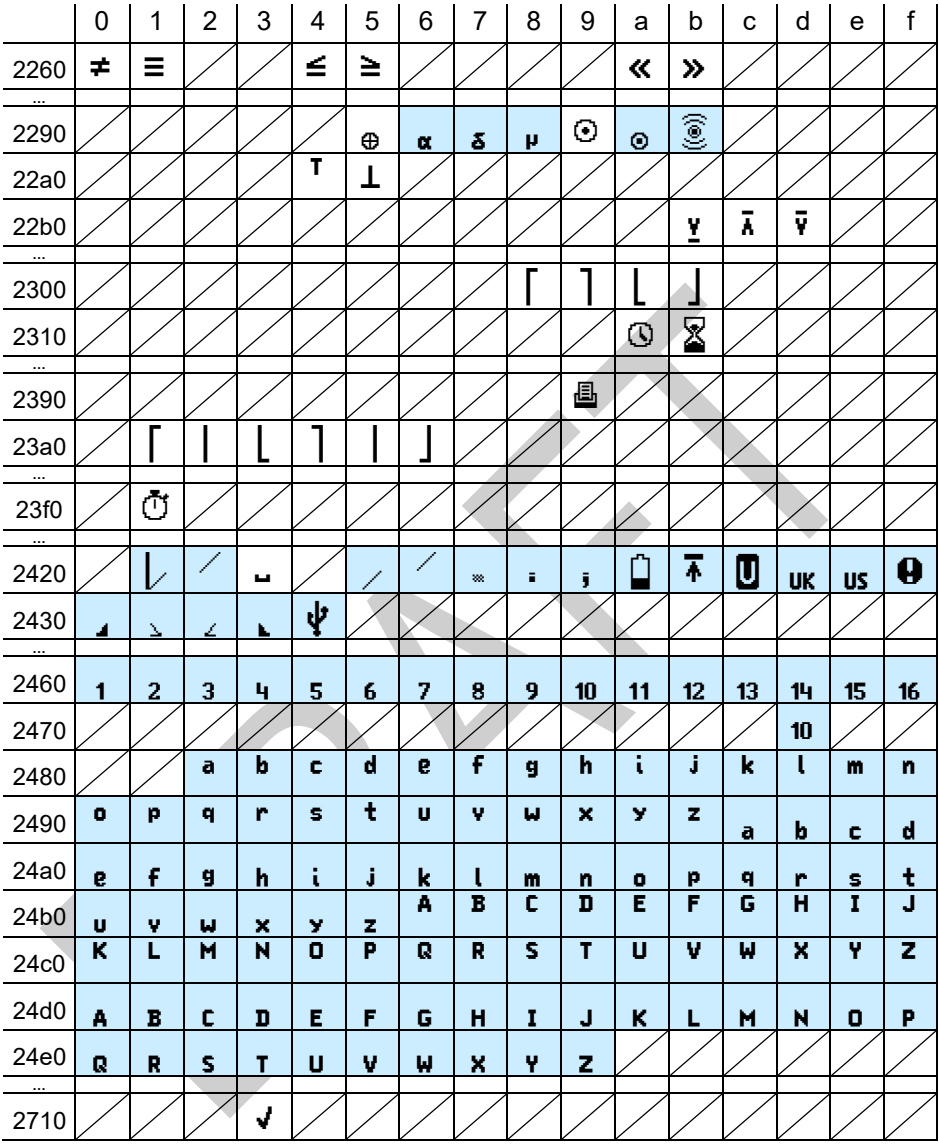
	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
24a0	ₑ				ᵢ	ⱼ	ₖ	ₗ	ₘ	ₙ	ₒ	ₚ	ᵩ		ₛ	
24b0	ᵤ	ᵥ	ₓ	ₓ	ᵧ	ᵤ										
24c0										T						
24d0	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
24e0	Q	R	S	T	U	V	W	X	Y	Z						

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0020		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
0030	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
0040	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
0050	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
0060		a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
0070	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	
...																
00a0		¡	¢	£		¥		§		ª		«	¬			
00b0	°	±				µ		·				»	¼	½		¿
00c0	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
00d0	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý		ß
00e0	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
00f0	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý		ÿ
0100	Ā	ā	Ă	ă	Ą	ą	Ć	ć					Č	č	Ď	ď
0110	Đ	đ	Ē	ē	Ě	ě	Ė	ė	Ę	ę	Ě	ě			Ğ	ğ
0120								ħ			Ī	ī	Ĭ	ĭ	Į	į
0130	İ	ı								Ĺ	Í			Ľ	ľ	
0140		Ł	ł	Ń	ń			Ň	ň				Ō	ō	Ŏ	ŏ
0150			Œ	œ	Ŕ	ŕ			Ř	ř	Ś	ś			Ş	ş
0160	Š	š	Ţ	ţ	Ť	ť			Ũ	ũ	Ū	ū	Ŭ	ŭ	Ů	ů
0170			Ų	ų	Ŵ	ŵ	Ŷ	Ŷ	Ÿ	Ź	ź	Ż	ż	Ž	ž	
...																
0230			y̅	y̅												
...																

DRAFT

	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0370									x̄	x̂						x̄
...																
0390	ΐ	Α	Β	Γ	Δ	Ε	Ζ	Η	Θ	Ι	Κ	Λ	Μ	Ν	Ξ	Ο
03a0	Π	Ρ		Σ	Τ	Υ	Φ	Χ	Ψ	Ω	Ϊ	Ϋ	ά	έ	ή	ί
03b0	ΰ	α	β	γ	δ	ε	ζ	η	ϑ	ι	κ	λ	μ	ν	ξ	ο
03c0	π	ρ	ς	σ	τ	υ	φ	χ	ψ	ω	ϊ	ϋ	ό	ύ	ώ	
...																
2000																
2010									‘	’	‚	‘	“	”	„	“
2020							…	1								
...																
2080	0	1	2	3	4	5	6	7	8	9	+	−				*
2090															∞	∞
20a0													€			
...																
2100			ℂ												h	ħ
2110				ℓ		ℕ					ℚ			ℝ		
2120					ℤ											
...																
2140								*e*	*i*							
...																
2160	0	1	2	3	4	5	6	7	8	9	+	−				
...																
2190	←	↑	→	↓		↕										
...																
21c0	→				⇕					Δ		∇	≡	↶		
...																
21e0								⇑		⇓						
...																
2200	∀	∁	∂	∃	∄	∅	Δ	∇	∈	∉		∋	∌		.	∏
2210				∓					∘	•	√	∛	∜	∝	∞	∟
2220	∠	∡	∢	∣	∤	∥	∦	∧	∨	∩	∪	∫	∬	∭	∮	∯
2230	∰						:									
2240				≃					≈							
2250					≔				≡	≙						

DRAFT



Complex Notation, Storage, and Calculations Close to Infinity

Like with *angles* or *short integers*, there are different ways a *complex number* can be written: either in *Cartesian* or polar notation, the latter with all kinds of angular units. As long as you stay away from infinities, every notation will do.

For reasons of tradition, rectangular notation is seen most frequently. We used it for storing *complex numbers* in the *WP 34S*. Thus, it is used for the *WP43* as well, but care must be taken at *complex* infinities.

Since infinities are counted as numeric data being part of the *real number* range (cf. p. 194), also *complex* infinities may be part of the *complex number* plane the *WP43* operates on. Coming from rectangular notation, eight '*complex* infinities' are possible only, listed here aside to their equivalents in polar notation (note the phase is counted counter-clockwise, starting with $\varphi = 0$ at the positive *real* axis):

Re(z)	Im(z)	r(z)	φ(z)	
$-\infty$	$-\infty$	∞	−135°	$-3\pi/4$
0	$-\infty$		−90°	$-\pi/2$
∞	$-\infty$		−45°	$-\pi/4$
∞	0		0°	0
∞	∞		45°	$\pi/4$
0	∞		90°	$\pi/2$
$-\infty$	∞		135°	$3\pi/4$
$-\infty$	0		180°	π

Calculating with *complex* infinities, every finite number may be neglected in comparison; so for $|\boldsymbol{x}| \neq \infty$ any inputs like, e.g., $\boldsymbol{x} + \boldsymbol{i} \times \infty$ may be replaced by $0 + \boldsymbol{i} \times \infty$, easing calculations significantly. Actually, you have to deal with just these eight cases listed above as long as you build your *complex* calculations on *Cartesian* notation (but cf. f. 132 on p. 200). With polar notation, on the other hand, an infinite number of *complex* infinities would have to be treated.

Note, however, that polar notation is advantageous for *complex* powers and roots: the $\boldsymbol{n}^{\text{th}}$ root of a *complex number* (*r*; *φ*) in polar notation will return $(\sqrt[n]{r}\,;\ \varphi/n)$. Hence, e.g., $\sqrt[3]{(\infty\,;\ 180°)} = (\infty\,;\ 60°)$, corresponding to the *Cartesian* point $\infty \times (1 + i\sqrt{3})$ which the calculator will display as $\infty + i \times \infty$, converted following the calculation rules – but mathematically wrong; and $\sqrt[6]{(\infty\,;\ 180°)} = (\infty\,;\ 30°)$, corresponding to the *Cartesian* point $\infty \times (\sqrt{3} + i)$, would return $\infty + i \times \infty$ as well.

Integer powers of (*r*; *φ*), on the other hand, will return $(r^n\,;\ \varphi \times n)$. E.g., $\infty + i \times \infty = (\infty\,;\ 45°)$ squared shall return $(\infty\,;\ 90°) = 0 + i\,\infty$. Naïve approach: $(\infty + i \times \infty)(\infty + i \times \infty) = \infty - \infty + 2\,i\,\infty = 0 + i\,\infty$ q.e.d. Note the two infinities are exactly identical here; but $\infty - \infty$ is generally defined as **NaN** for good reasons, so the properly calculated result returned will deviate from the truth here, too.

Thus, the odds are high that roots and powers of *complex numbers* near the

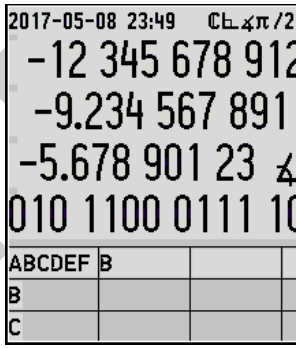
DRAFT

edge of the *complex* plane will return incorrect data, in particular if entered in polar notation (and translated internally to *Cartesian*). These errors seem to be inevitable.

Display Segmentation

The *LCD* of the *WP43* is full dot matrix (cf. *App. A*). Top down, you will find

1. 20 px for the *status bar*,
2. 4 blank pixel rows for separation,
3. 147 px for either
   a. the contents of up to 4 *stack registers*, or
   b. 7 program steps in *PEM*, or
   c. up to 7 rows of numeric output of SHOW, and
4. 69 px maximum for the menu section.



Reasons for these figures are:

- Each regular alphanumeric row (e.g., for labels, status, a text string, or a program step) requires 20 px vertically (cf. pp. B-8f). There shall be at least one row of blank pixels separating it from the next row of data.
- Hence, each *menu* row takes (counting bottom-up) 1 + 20 + 1 + 1 = 23 px (the first pixel is for separation from the black frame, the last for its upper frame line). Thus, three *menu* rows require 69 px. In U→, extra-high labels may appear: they will require 1 + 20 + 1 + 20 + 1 + 1 = 44 px for double height or even 1 + 20 + 1 + 20 + 1 + 20 + 1 + 1 = 65 px for triple height then.
- At top of the *LCD*, the *status bar* takes another 20 px plus 4 for separation. Thus, 240 − 69 − 24 = 147 px remain free on the screen so far.
- Each regular numeric output requires 32 px vertically (cf. p. B-8) plus 4 px separating it from the next such output row. Thus, for 4 rows we need 4 × 32 + 3 × 4 = 140 px. Since there are 4 px above them already, we put the remaining 7 px below this block.
- If there is a short *text string* in an arbitrary *stack register*, its base line should be positioned where the base line of the respective numeric output would have been.
- With a *text string* in **X** needing two rows, 1 + 20 + 1 + 20 + 1 = 43 px are required vertically matching the 7 + 32 + 4 = 43 px available for the lowest numeric row. Longer *strings* will need SHOW to be shown entirely.

DRAFT

- In *PEM*, on the other hand, 147 = 7 × (20 + 1) px correspond to a block of 7 alphanumeric program rows.
- With SHOW, also pure numeric output may require more than one row – the small font will be used there as well. Thus, up to 7 rows of small digits are possible again. Cf. p. 83.

In the *menu section*, we also have a horizontal structure for the six *menu* keys. We start one pixel off the black frame at left display edge. On the right edge, the *characters* themselves contain at least one blank column. A minimum of 2 px separate *softkey* labels from each other (one black and one blank). This way we lose a total of 1 + 5 × 2 px. The remaining 389 px mean a width of 65 available for 6 *softkey* labels, corresponding to six standard width letters (though letters may extend from 4 to 14 px in small font) which should be centered as good as possible. Note that labels in *menu views* may be not fully displayed if they are wider than 64 px, so labels deviating only in their very last *characters* may become visually indistinguishable. Users should avoid such ambiguities.

If gaps are chosen between groups of digits (GAP ≠ 0), they are filled by a punctuation space each.

Echo and Fallback

Almost all key presses are echoed and fall back to NOP. *Softkeys* shall not since *WYSIWYG* applies here always. Furthermore, some functionalities, wherever assigned to, are never echoed (and hence can't fall back) since

1) they are harmless (EXIT, UP, DOWN) or
2) reverting or exiting them is a no-brainer (UP, DOWN, P/R, all *menu* calls) requiring just one keystroke.

With the presence of UNDO (→ B-28), fallback to NOP becomes debatable overall; there is a benefit remaining in *user mode* as long as an overlay matching the actual assignments is not available – and UNDO is shifted in *SDC*.

Equations

Equations are entered in EQN as written (i.e. following algebraic notation and rules). While editing them, punctuation spaces are automatically inserted after each constant or *variable* (a *variable name* ends when the next operator or a **:** or a **)** is entered) as well as after **=** and behind each operator like **+**, **−**, **·**, **×**, **/**, and **!** (except **^**); a punctuation space is inserted after **:**. There is no implicit multiplication.

DRAFT

Other functions like roots, absolute values, or trigs shall be written using parentheses or bars, e.g.,

- pressing [ ] [√x] shall display √(∞),
- pressing [ ] [e^x] shall display e^(∞), and
- pressing [|x|] shall display |∞|;
- for cosine either pick it from CATALOG'FCNS or enter cos() – both will be echoed as cos(∞);

then specify the argument. The same applies to dyadic (like COMB) and triadic operations in analogy – their arguments shall be separated by colons inserted from the *Equation Editor menu*.

Terminating the *Equation Editor*, numeric exponents are automatically converted from, e.g., xy ^23 to xy^{23}. For easier handling, this will be reverted when editing such an equation again.

Execution Times

The time a particular command takes to be executed depends on the command and its argument or parameter. For trigonometric functions, for example, execution times may easily vary by a factor of two.

Just for your information, here are measurement results for some monadic functions operating on $x = \pi/6.3 \approx 0.498\,665\,5$ on a *WP43* v0.27 pilot powered through *USB*:

erfc	67 ms	
Γ(x)	45 ms	
lnΓ	37 ms	
arcosh	22,2 ms	
erf	21,8 ms	
sinc	12.4 ms	in RAD
E(m), K(m)	10,8 ms	
$(-1)^x$	6.6 ms	in RAD
sincπ	6.1 ms	in RAD
tanh, arccos	5.6 ms	
arsinh	4,9 ms	
e^x, sinh, cosh	4.8 ms	
arcsin, arctan	4.6 ms	

DRAFT

2^x	4.5 ms	
10^x, artanh	3.9 ms	
tan	3.8 ms	in RAD
ln, lg, lb	3,2 ms	
$\sqrt[3]{x}$	2,4 ms	
sin, cos	2.3 ms	in RAD[190]
$\sqrt{(1+x^2)}$	1.8 ms	
$\sqrt{x}$	1.3 ms	
x^3	0.7 ms	
1/x, x^2, FP, IP, ceil, floor, ROUNDI, |x|	0,4 ms	… 0.5 ms

Functionality of ENTER↑

ENTER↑ on the *WP43* works as it does on the *HP-42S* – our reference – and on our *WP 34S* and *31S*. There were some other *HP RPN* pocket calculators featuring slight variants of this functionality. See this compilation by *Richard Nelson* for more:
http://h20331.www2.hp.com/hpsub/downloads/S07%20HP%20RPN%20Evolves%20V5b.pdf

Names of User Defined Items

Such a *name* may consist of max. 7 *characters* containing at least 1 letter, and it shall begin with a letter. For user-created *labels*, *menus*, and *variables*, +, −, ×, /, :, ^, (,), =, !, ;, |, comma, and space must not be used.

Examples of legal user-defined *names* are **where?**, **all.in**, and **Go_17** (cf. also p. 334).

Predefined commands, constants, *menus*, *variables*, and *system flags* do not suffer from these restrictions.

Number Range

A number range up to 10^{99} is sufficient for almost all real-world problems – else common scientific calculators would generally feature a larger numeric range (cf. also p. 125). So we can conclude that the *real number* range supported by your *WP43* (cf. p. 190) suffices by far for solving what has to be solved. Saving

[190] For the *angle* used here, computing the sine in DEG took 0.1 ms less.

DRAFT

display space for sensible results gave reason for RANGE.

For sake of coherence, maximum numbers within different *DTs* should match. I.e. the maximum absolute value allowed for a *long integer* should be approximately equal to the respective values for a *real* and the parts of a *complex number*. Theoretically. In real life, huge *angles* are meaningless.

Note the number range determines the precision required for calculating accurately (cf. p. B-8).

Precision and Accuracy

As mentioned above more than once, there are inevitable errors in each numeric calculation step, frequently caused by rounding to the internal finite precision featured by the calculator. Already a simple fraction like $1/3$ (stored as a *real number* following the representation as explained on pp. 191ff) deviates from the truth by more than 3×10^{-35}. This looks very small though such errors will accumulate during longer calculations (cf. pp. 298f).

In the check on p. 262 using primes, results ≠ 0. happen more frequently between 700 and 1000, while you won't find a single one between 1000 and 5300!

In the majority of cases we calculate with 39 internal digits and round the result to 34 digits. If true then...

- **3** [1/x] **3** [×] will return 0.333 333 333 333 333 333 333 333 333 333 333 3 × 3 = 0.999 999 999 999 999 999 999 999 999 999 9. This is the exact 34-digit product.
- **0** [CC] **1** [√x] will return 0.707 106 781 186 547 524 400 844 362 104 849 0*39 2*... + ***i*** × 0.707 106 781 186 547 524 400 844 362 104 849 0*39 2*...

  Squaring this should return
  $(0.707\,106\,781\,186\,547\,524\,400\,844\,362\,104\,849\,0)^2 -$
  $(0.707\,106\,781\,186\,547\,524\,400\,844\,362\,104\,849\,0)^2 +$
  $2 \times i \times (0.707\,106\,781\,186\,547\,524\,400\,844\,362\,104\,849\,0)^2$
  $= 0 + 2 \times i \times 0.499\,999\,999\,999\,999\,999\,999\,999\,999\,999\,999\,9\mathit{44\,4}...$
  $= i \times 0.999\,999\,999\,999\,999\,999\,999\,999\,999\,999\,999\,8\mathit{88\,8}...$
  $= i \times 0.999\,999\,999\,999\,999\,999\,999\,999\,999\,999\,999\,9$. This is the best 34-digit result possible. If the result was 1×10^{-34} greater, it would round to 1.0.

In real-world problems, usually the least accurate of all input parameters determines the accuracy of the result. In the standard test mentioned on p. 262 starting with 9°, you can nevertheless get 28-digits precision in the result since the input of 9° is exact (but note one digit precision is lost with each step a trigonometric function is calculated there).

DRAFT

Internally, e.g., the *WP 34S* computes with 39 digits and rounds the results to 34 or 16 digits, respectively. Consequently, *WP43* also works with 39 digits internally most times and rounds results to 34 digits. SLVQ calculates with 72 digits. Each of the 79 constants in CONST as well as the 62 conversion factors and most of the about 200 auxiliary constants (like ln(2), ln(10), $\pi/2$, $\pi/4$, $3\pi/4$, $\frac{1}{2}\sqrt{2}$, $\frac{1}{2}\sqrt{3}$) feature 55 significant digits internally (although many of these digits may be zero – see `generateConstants.c`; there are 368 constants in total). The statistical summation registers are 75 digits wide (used also for the initial steps of variance calculation). Range reductions for trigonometric functions use many more digits (cf. pp. B-7f).

Real-world problems are usually many orders of magnitude less precisely specified than the internal precision of your *WP43*. Compare also the precisions in the set of physical and astronomical constants provided on pp. 113ff. To get an idea what an 'order of magnitude' means in real world, look there as well. Cf. also f. 73 on p. 124.

Prefixes

Prefixes f and g passed without any discussion for more than six years until 2019-06. Alternatively, prefixes ↰ and ↱ could have been chosen but their typography leaves less freedom for placing the corresponding golden and blue labels. Dropping the letters **f** and **g** – resulting in ▭ and ▭ – made b/w copies of the manuals less useful.

Random Numbers

RAN# uses a *PCG* random generator (see *http://www.pcg-random.org*).

RPN vs. RPL or Vice Versa

HP moved from *RPN* to *RPL* in 1986 for their high-end scientific pocket calculators. There were good reasons for doing this move. *RPL* is a very coherent operating system – actually it is so coherent that it hurts (in my opinion). It carries the postfix principle to the extreme.

RPL may be next to perfect (for programming, in particular) if you use it every day (like *SAP* in business) – else it is annoying, however (also like *SAP*). Since almost all of our users use their *WP43* casually nowadays, the annoying part would become overwhelming very soon. Thus, we stick to classic *RPN* – quick and dirty, but far easier to resume after pausing for days, weeks, or even months.

DRAFT

It took me until 2025 to realize that we had silently taken over one of the best properties of *RPL*: object orientation.[191] By introducing *data types* and letting functions operate the same way on objects of various *DTs*, we eased computing significantly compared to, e.g., *HP-41C* or *HP-15C*. I didn't notice that earlier since our reference, the *HP-42S*, used *DTs* already though to less extent. Actually, one should discern *RPN* before *RPL* from *RPN* thereafter.

Presently, there are projects called *DB48X* and *NewRPL* pursuing an extended *RPL* implementation on a new platform (since *HP* abandoned *RPL* ceasing production of the *HP-50g* in 2015). There is, however, no calculator hardware readily available to host this system as far as I am aware of.

Sorting in Detail

There is no international standard for sorting *characters*; we had to invent our own. Thus, sorting of *items* (i.e. commands, *variables*, *text strings*, *labels*, *system flags*, etc.) on *WP43* works as listed below, top down and left to right. Note that sorting is a two-step procedure: step 1 sorts the text strings under consideration just according to column 1 of this table, comparing them; if two strings are rated equal in this aspect, step 2 takes the columns following into account. The *IOI* is sorted according to this algorithm (hopefully), CATALOG is definitely.

Sorting is specified for the small font below. It holds also for the large font as far as *characters* are applicable there. The 4-digit number following each *character* in the table is its hexadecimal *Unicode*. 163 codes printed on blue were abused for *characters* we needed but did not find in the range we covered. *Characters* printed on bright yellow or orange are neither accessible through the keyboard nor contained in any alpha *menu*; the orange ones are designed but not used at all by *WP43* so far.

␣ 0020	2003	2004	…	2008	200a	␣ 2423
0 0030	∎ 220e	° 00b0	^{0} 2160	$_0$ 2080		
1 0031	1 2027	¼ 00bc	½ 00bd	^{1} 2161	$_1$ 2081	1 2460
2 0032	^{2} 2162	$_2$ 2082	2 2461			
3 0033	^{3} 2163	$_3$ 2083	3 2462	∛ 221b		
4 0034	^{4} 2164	$_4$ 2084	4 2463			
5 0035	^{5} 2165	$_5$ 2085	5 2464			

[191] Thanks to *c3d* for pointing me to that.

6 0036	6 2166	6 2086	6 2465			
7 0037	7 2167	7 2087	7 2466			
8 0038	8 2168	8 2088	8 2467			
9 0039	9 2169	9 2089	9 2468			
10 247d	10 2469					
11 246a						
12 246b						
13 246c						
14 246d						
15 246e						
16 246f						
A 0041	a 0061	A 24b6	a 2482	A 24d0	a 249c	
	À 00c0	à 00e0	Á 00c1	á 00e1	Â 00c2	â 00e2
	Ã 00c3	ã 00e3	Ä 00c4	ä 00e4	Å 00c5	å 00e5
	Æ 00c6	æ 00e6	Ā 0100	ā 0101	Ă 0102	ă 0103
					Ą 0104	ą 0105
B 0042	b 0062	B 24b7	b 2483	B 24d1	b 249d	
C 0043	c 0063	C 24b8	c 2484	C 24d2	c 249e	
	Ç 00c7	ç 00e7	Ć 0106	ć 0107	Č 010c	č 010d
					Ⅽ 2102	C 2201
D 0044	d 0064	D 24b9	d 2485	D 24d3	d 249f	
	Ð 00d0	ð 00f0	Ď 010e	ď 010f	Đ 0110	đ 0111
E 0045	e 0065	E 24ba	e 2486	E 24d4	e 24a0	È 00c8
	è 00e8	É 00c9	é 00e9	Ê 00ca	ê 00ea	Ë 00cb
	ë 00eb	Ē 0112	ē 0113	Ĕ 0114	ĕ 0115	Ė 0116
	ė 0117	Ę 0118	ę 0119	Ě 011a	ě 011b	E 2073
	e 2147					
F 0046	f 0066	F 24bb	f 2486	F 24d5	f 24a1	
G 0047	g 0067	G 24bc	f 2487	G 24d6	g 24a2	
					Ğ 011e	ğ 011f

DRAFT

H 0048	h 0068	h 210e	ᴴ 24bd	ₕ 2488	ʰ 24d7	ₕ 24a3
					ħ 0127	ℏ 210f
I 0049	i 0069	ᴵ 24be	ᵢ 2489	ᵢ 24d8	ⁱ 24a4	
	Ì 00cc	ì 00ec	Í 00cd	í 00ed	Î 00ce	î 00ee
	Ï 00cf	ï 00ef	Ī 012a	ī 012b	Ǐ 012c	ǐ 012d
	Į 012e	į 012f	İ 0130	ı 0131	ⅈ 2148	
J 004a	j 006a	ᴶ 24bf	ʲ 248a	ⱼ 24d9	ʲ 24a5	
K 004b	k 006b	ᴷ 24c0	ₖ 248b	ₖ 24da	ᵏ 24a6	
L 004c	l 006c	ᴸ 24c1	ₗ 248c	ₗ 24db	ˡ 24a7	ℓ 2113
	Ĺ 0139	Ɩ 013a	Ľ 013d	ľ 013e	Ł 0141	ł 0142
M 004d	m 006d	ᴹ 24c2	ₘ 248d	ₘ 24dc	ᵐ 24a8	
N 004e	n 006e	ᴺ 24c3	ₙ 248e	ₙ 24dd	ⁿ 24a9	Ñ 00d1
	ñ 00f1	Ń 0143	ń 0144	Ň 0147	ň 0148	ℕ 2115
O 004f	o 006f	ᴼ 24c4	ₒ 248f	ℴ 00a9	ᵒ 24de	ₒ 24aa
	Ò 00d2	ò 00f2	Ó 00d3	ó 00f3	Ô 00d4	ô 00f4
	Õ 00d5	õ 00f5	Ö 00d6	ö 00f6	Ø 00d8	ø 00f8
	Ō 014c	ō 014d	Ŏ 014e	ŏ 014f	Œ 0152	œ 0153
P 0050	p 0070	ᴾ 24c5	ₚ 2490	ₚ 24df	ᵖ 24ab	
Q 0051	q 0071	ᵠ 24c6	ₑ 2491	ᵩ 24e0	ᵠ 24ac	ℚ 211a
R 0052	r 0072	ᴿ 24c7	ᵣ 2492	ᵣ 24e1	ʳ 24ad	
		Ŕ 0154	ŕ 0155	Ř 0158	ř 0159	ℝ 211d
S 0053	s 0073	ˢ 24c8	ₛ 2493	ₛ 24e2	ˢ 24ae	Ś 015a
	ś 015b	Ş 015e	ş 015f	Š 0160	š 0161	ß 00df
T 0054	t 0074	ᵀ 24c9	ₜ 22a4	ₜ 2494	ᵀ 24e3	ₜ 24af
			Ţ 0162	ţ 0163	Ť 0164	ť 0165
U 0055	u 0075	ᵁ 24ca	ᵤ 2495	ᵤ 24e4	ᵘ 24b0	
	Ù 00d9	ù 00f9	Ú 00da	ú 00fa	Û 00db	û 00fb
	Ü 00dc	ü 00fc	Ũ 0168	ũ 0169	Ū 016a	ū 016b
	Ŭ 016c	ŭ 016d	Ů 016e	ů 016f	Ų 0172	ų 0173
V 0056	v 0076	ⱽ 24cb	ᵥ 2496	ᵥ 24e5	ᵛ 24b1	

DRAFT

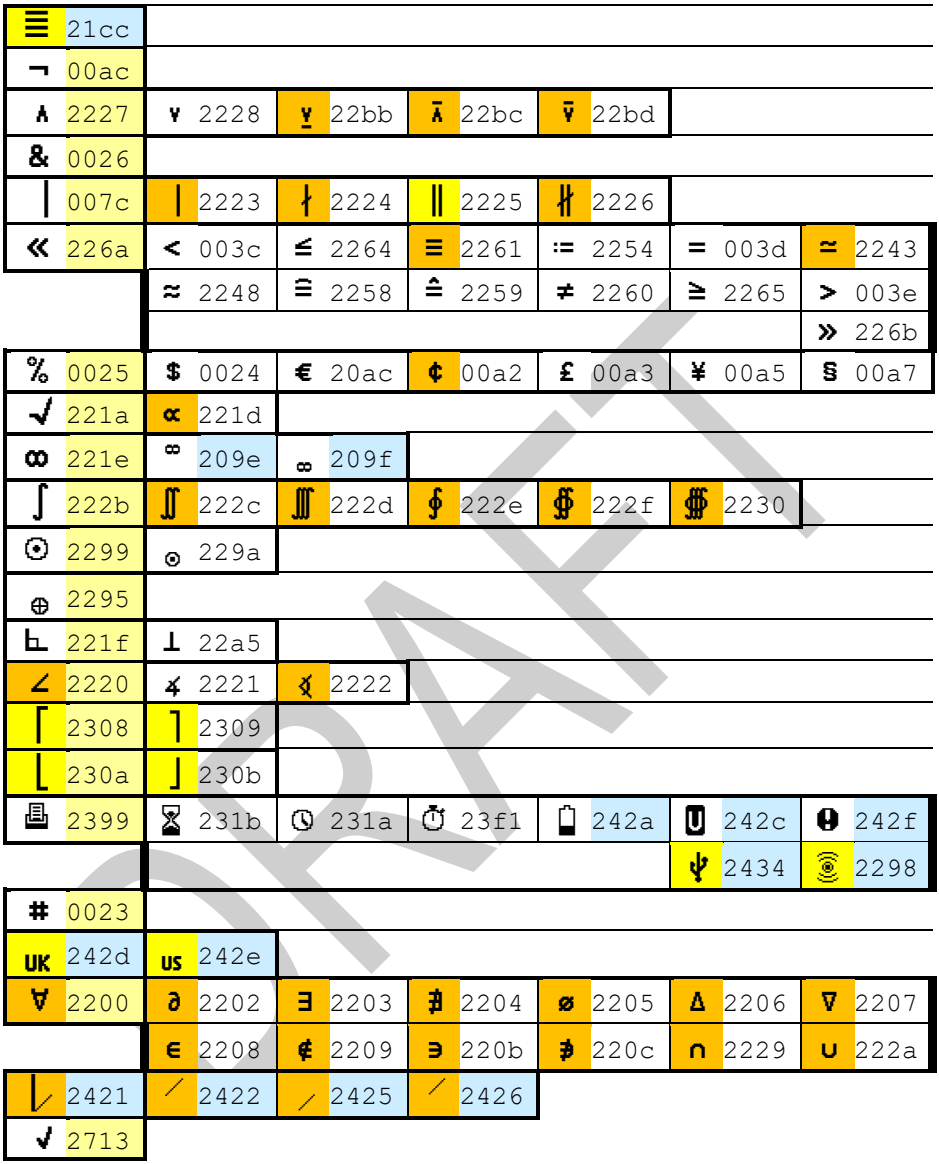
W 0057	ω 0077	ʷ 24cc	ᵚ 2497	ₙ 24e6	ₒ 24b2	
					Ŵ 0174	ŵ 0175
X 0058	x 0078	ˣ 24cd	ᵡ 2498	ₓ 24e7	ₓ 24b3	
			x̂ 0379	x̄ 0378	x̄ 037f	x⃰ 221c
Y 0059	y 0079	ʸ 24ce	ᵞ 2499	ᵧ 24e8	ᵧ 24b4	
	Ý 00dd	ý 00fd	Ŷ 0176	ŷ 0177	Ÿ 0178	ÿ 00ff
					ȳ 0233	ȳ 0232
Z 005a	z 007a	ᶻ 24cf	ᶻ 249a	ᵤ 24e9	ᵤ 24b5	Ź 0179
	ź 017a	Ż 017b	ż 017c	Ž 017d	ž 017e	ℤ 2124
A 0391	α 03b1	ₐ 2296	ά 03ac			
B 0392	β 03b2					
Γ 0393	γ 03b3					
Δ 0394	δ 03b4	ₔ 2297				
E 0395	ε 03b5	έ 03ad				
Z 0396	ζ 03b6					
H 0397	η 03b7	ή 03ae				
Θ 0398	ϑ 03b8					
I 0399	ι 03b9	ί 03af	Ϊ 03aa	ϊ 03ca	ΐ 0390	
K 039a	κ 03ba					
Λ 039b	λ 03bb					
M 039c	μ 03bc	µ 00b5	ₘ 2298			
N 039d	ν 03bd					
Ξ 039e	ξ 03be					
O 039f	ο 03bf	ό 03cc				
Π 03a0	ϖ 220f	π 03c0				
P 03a1	ρ 03c1					
Σ 03a3	σ 03c3	ς 03c2				
T 03a4	τ 03c4					
Y 03a5	υ 03c5	ύ 03cd	Ϋ 03ab	ϋ 03cb	ΰ 03b0	
Φ 03a6	φ 03c6					
X 03a7	χ 03c7					

DRAFT

Ψ 03a8	ψ 03c8					
Ω 03a9	ω 03c9	ώ 03ce				
(0028	) 0029					
[005b	⌈ 23a1	⎢ 23a2	⌊ 23a3			
			] 005d	⌉ 23a4	⎥ 23a5	⌋ 23a6
{ 007b	} 007d					
◢ 2430	◣ 2431	◤ 2432	◥ 2433			
+ 002b	₊ 208a	⁺ 216a	± 00b1			
- 002d	₋ 208b	⁻¹ 2072	⁻ 216b	∓ 2213		
× 00d7	· 00b7	• 2219	∘ 2218	∗ 002a	⁎ 208f	
/ 002f	\ 005c					
^ 005e						
, 002c	; 2429					
. 002e	⁝ 2428	... 2026				
! 0021	¡ 00a1					
? 003f	¿ 00bf					
: 003a	: 2236	÷ 00f7				
; 003b						
' 0027	‘ 2018	’ 2019	‚ 201a	‛ 201b	[192]	
" 0022	“ 201c	” 201d	„ 201e	‟ 201f	« 00ab	» 00bb
@ 0040						
_ 005f	⍽ 2427					
~ 007e						
→ 2192	➔ 21c0					
← 2190	↶ 21cd					
↑ 2191	⇧ 21e7	⍋ 21c9	⍒ 242b			
↓ 2193	⇩ 21e9	⍒ 21cb				
⇄ 21c4						
⇕ 2195						

[192] Please look up https://de.wikipedia.org/wiki/Anführungszeichen#Andere_Sprachen or https://en.wikipedia.org/wiki/Quotation_mark#Summary_table for properly using these *characters* in different languages.

DRAFT



Stack Lift Disabling Functions

For sake of backward compatibility, we decided keeping these as they were on the vintage *HP RPN* pocket calculators up to the *HP-42S* (and *WP 34S* and *31S*):

DRAFT

Only ENTER↑, CLX, Σ+, Σ−, M.EDI, M.EDIN, M.←, M.↑, M.→, and M.↓ disable *ASL*, all other functions enable it. But compare INPUT on p. 48.

Stack Size

In 2013, *stack* size was discussed for this project. An *RPL*-like 'infinite' *stack* would allow for saving (pushing) everything thereon before calling a (sub-) routine, and popping it after RTN; though it makes traditional R↓, R↑, and top level repetition obsolete (and FILL as well). In this context I suggested two new commands called CLOSES and OPENS for closing the bottom section (4 or 8 *registers*) of an infinite *stack* for the time when R↓, R↑, FILL, and top level repetition were required, and opening it thereafter.

At the bottom line, however, 8 *stack registers* turned out being sufficient for solving every real-world mathematical, scientific, or engineering problem (cf. *OMS 1* as well as field experience with *WP 34S* and *31S* since 2011). So we decided to stick to classic *RPN* as implemented on the *WP 34S* and *31S*: it simply covers everything needed most easily. For special requirements, STOSTK and RCLSTK are provided.

Stack overflow is another topic reappearing every once and a while. Principally, it can happen after entering the 3rd level in a traditional 4-*register stack*:

					1	1	2
			1	1	2	2	3
	1	1	2	2	3	3	4
1_	1	2_	2	3_	3	4_	4
1	ENTER↑	**2**	ENTER↑	**3**	ENTER↑	**4**	ENTER↑

Starting with an empty *stack*, you could watch T and abort ENTER↑ when $t \neq 0$. Though this will work the very first time only since *t* will never become 0. again but you will want to continue calculating. Setting SSIZE8 is easier – from then on you can calculate just *sans soucis*.

Structured Programming

In 2013, I suggested the following control structures:

- IF … THEN … ELSE … END,
- FOR … FROM … TO … END,
- REPEAT … UNTIL, and

DRAFT

- WHILE … END.

Traditional END would need to be called ENDPGM then.

Later, we discussed some *PASCAL*-like structures:

- IF … THEN BEGIN … END ELSE BEGIN … END;
- FOR … DO BEGIN … END; and
- WHILE … DO BEGIN … END;

Eventually, we refrained from implementing such commands since we had doubts about the sensibility of mixing keystroke and structured programming paradigms.

UNDO

In 2013, UNDO was planned for *WP43* as it works in *HP-48*, recalling just the *stack* as it was before executing last command. The *WP 31S* of 2014, on the other hand, features an UNDO recalling the entire *calculator state* as it was before. Until 2020, we assumed that such a complete UNDO was viable in *WP43* as well, but the necessary overhead turned out forbidding.

Since

- any *user flag* altered inadvertently can be reverted easily and
- any wide-reaching clear command (CLREGS, CLFALL, etc.) asks for confirmation before executing,

we implemented UNDO recalling not only the *stack* but also the statistics storage (i.e. STATS) and all *system flags* as they were before executing the last command, for better user experience (note this specification also allows for undoing Σ−).

> While we are at it, undoing the previous program step while going back in program memory (via BST) is not viable. It would require saving also the previous content of the *registers* and *variables* for each operation affecting them. Think of undoing delete or clear operations, for instance.

It turned out that undoing ENTER↑ and EXIT can be ambiguous. The *WP 31S* could not serve as model since the functionality of EXIT there deviates from the one on *HP-42S* and *WP43*. Here, we specified these functionalities so that [ENTER↑] [↶] equals EXIT and [EXIT] [↶] equals NOP (but with ***x*** being closed):

DRAFT

1	1 1	1 ABC	1 ABC ABC	1 ABC	1 ABC 123
1	ENTER	α **ABC**	ENTER	↶	**123**

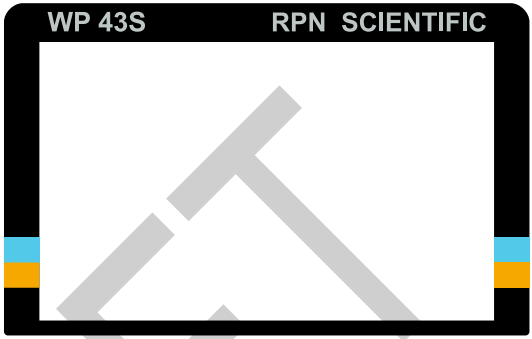
1	1 1	1 2	1 2 2	1 2	1 2 123
1	ENTER	**2**	ENTER	↶	**123**

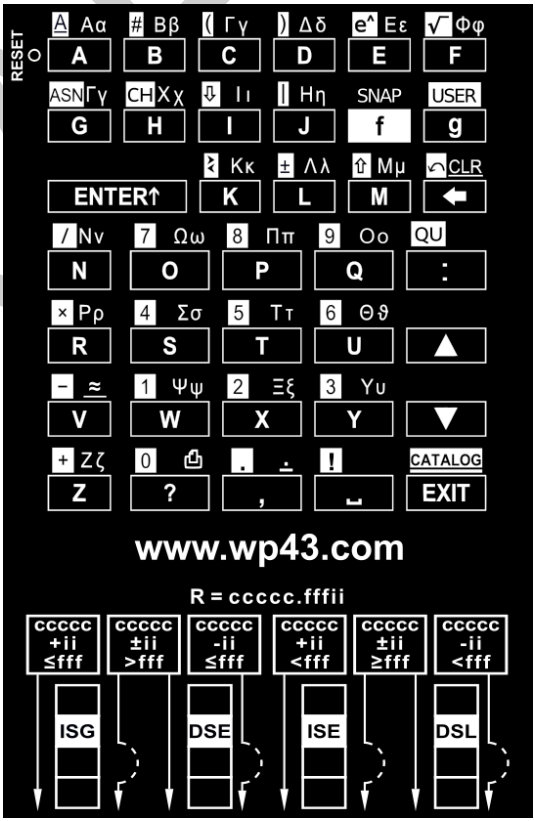
1	1	1 2	1 2	1 2	1 2 123
1	EXIT	**2**	EXIT	↶	**123**

DRAFT

FOR YOUR CONVENIENCE

Please find here the display frame for your *WP43* as we designed it (printed to scale) and below its virtual keyboard in *alpha input mode* (*AIM*) plus a branching helper. Overleaf, the original *WP43* keyboard is printed to scale. If you bought your *WP43* off the shelf, keys and bezel should be printed properly (except shifted 7 and 🟨-shifted 8 which were not yet populated then), and the b/w print shown bottom right here shall be found laser-engraved on its rear for your reference (almost – we changed a few positions since 2022-10).





Alternatively, install *GIMP* or *INKSCAPE*, then look in *https://gitlab.com/rpncalculators/wp43/-/tree/master/res/artwork* for the file `WP43_layout.svg`; open it, hide all layers except "Background", "Overlay", "Bezel blue", "Bezel gold", and "Bezel grey". Then print, cut, and apply.

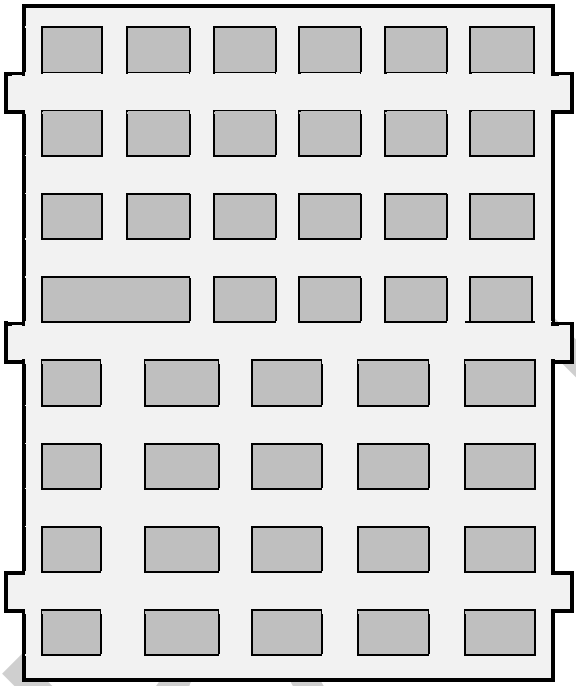
Furthermore, find overleaf an

DRAFT

explanatory picture for *short integer* operations taken from the back of an *HP-16C* (with the letters C denoting CARRY and G standing for OVERFL).

	C	G	
⊞ ⊟	x	x	
⊠	--	x	
÷	x	x	RMD≠0 → C
√x	x	--	RMD≠0 → C
CHS ABS	--	x	
DBL×	--	o	y · x → (X&Y)
DBL÷	x	o	(y&z) ÷ x → X ; RMD≠0 → C
SL	x	--	C ← o
SR	x	--	o → C
ASR	x	--	→ C
RL	x	--	C ←
RR	x	--	→ C
RLC	x	--	C ←
RRC	x	--	→ C

From the *USA*, here is a simple template for a quick and easy overlay (almost to scale – see point 3 below):



1. Print it on standard copy paper (80 $\mathrm{g/m^2}$). Fill in labels if, where, and as required.
2. Laminate with 80 $\mu\mathrm{m}$ stock (result is a thickness of about 0.25 mm).
3. Cut out the perimeter. Note the protruding tabs (a.k.a. tongues) as drawn here are too wide and too long (compare previous page); cut smaller. For the voids, a standard 6 mm square punch might be helpful (cf. a link on p. 184) since punches fitting our needs exactly will not be available off shelf.

DRAFT